Attorney's Docket No.: 460-010020-US(PAR)

2-20-01 PATENT 04CO #2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: COFTA et al.
Serial No.: 09/739,941
Filed: 12/18/00
For: METHOD FOR BINDING A PROGRAM MODULE

Group No.:

Examiner:

Commissioner of Patents and Trademarks
Washington, D.C. 20231

TRANSMITTAL OF CERTIFIED COPY

Attached please find the certified copy of the foreign application from which priority is claimed for this case:

Country	**: Finland**
Application Number	**: 19992786**
Filing Date	**: 27 December 1999**

WARNING: "When a document that is required by statute to be certified must be filed, a copy, including a photocopy or facsimile transmission of the certification is not acceptable." 37 CFR 1.4(f) (emphasis added.)

SIGNATURE OF ATTORNEY
Clarence A. Green

Type or print name of attorney

Reg. No.: 24,622

Tel. No.: (203) 259-1800

Perman & Green, LLP

Customer No.: 2512

P.O. Address

425 Post Road, Fairfield, CT 06430

NOTE: The claim to priority need be in no special form and may be made by the attorney or agent if the foreign application is referred to in the oath or declaration as required by § 1.63.

CERTIFICATE OF MAILING/TRANSMISSION (37 CFR 1.8a)

I hereby certify that this correspondence is, on the date shown below, being:

☒ MAILING

deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231

Date: 1/23/2001

☐ FACSIMILE

transmitted by facsimile to the Patent and Trademark Office

Deborah J. Clark
Signature

DEBORAH J. CLARK
(type or print name of person certifying)

PATENTTI- JA REKISTERIHALLITUS
NATIONAL BOARD OF PATENTS AND REGISTRATION

Helsinki 13.12.2000

E T U O I K E U S T O D I S T U S
P R I O R I T Y D O C U M E N T

Hakija Applicant	**Nokia Mobile Phones Ltd** **Espoo**
Patenttihakemus nro Patent application no	**19992786**
Tekemispäivä Filing date	**27.12.1999**
Kansainvälinen luokka International class	**G06F**

Keksinnön nimitys
Title of invention

"Menetelmä ohjelmamoduulin lataamiseksi"

Täten todistetaan, että oheiset asiakirjat ovat tarkkoja jäljennöksiä patentti- ja rekisterihallitukselle alkuaan annetuista selityksestä, patenttivaatimuksista, tiivistelmästä ja piirustuksista.

This is to certify that the annexed documents are true copies of the description, claims, abstract and drawings originally filed with the Finnish Patent Office.

Marketta Tehikoski
Apulaistarkastaja

Maksu 300,- mk
Fee 300,- FIM

Osoite: Arkadiankatu 6 A, P.O.Box 1160, FIN-00101 Helsinki, FINLAND
Puhelin: 09 6939 500
Telephone: + 358 9 6939 500
Telefax: 09 6939 5328
Telefax: + 358 9 6939 5328

A1

Menetelmä ohjelmamoduulin lataamiseksi

Nyt esillä oleva keksintö kohdistuu oheisen patenttivaatimuksen 1 johdanto-osan mukaiseen menetelmään ohjelmamoduulin lataamiseksi. Keksintö kohdistuu lisäksi oheisen patenttivaatimuksen 7 johdanto-osan mukaiseen päätelaitteeseen.

Ohjelmat toteutetaan nykyisin pääasiassa moduulirakenteisina, jolloin yksi ohjelma koostuu useista ohjelmamoduuleista, moduuleista, jne. Tällöin yksittäisen ohjelmamoduulin koko voidaan pitää rajallisena, ja käytön aikana ladataan vain sellaiset ohjelmamoduulit, joita kulloinkin tarvitaan. Tällainen järjestely helpottaa mm. ohjelmien ylläpitoa, pienentää ohjelmaa käyttävän tietojenkäsittelylaitteen muistitarvetta sekä mahdollistaa sen, että samaa ohjelmamoduulia voi käyttää useampi eri ohjelma. Tällaisen moduuleista koostuvan ohjelman suorittamiseksi käynnistetään ns. pääohjelma. Pääohjelman toiminnan edetessä esimerkiksi käyttäjien antamien ohjeiden perusteella pääohjelmasta ladataan kulloinkin tarvittava ohjelmamoduuli. Jatkossa tässä selityksessä käytetään yleisnimitystä ohjelmamoduuli sellaisesta ohjelmakomponentista, jonka yhteydessä latausta voidaan soveltaa. Tällaisia ohjelmamoduuleita ovat mm. ohjelmakirjastot.

Latauksella (Binding) tässä selityksessä tarkoitetaan ohjelmistojen suoritukseen liittyvää prosessia, jossa ohjelmakomponentti, kuten pääohjelma tai aliohjelma, kutsuu toista ohjelmakomponenttia, kuten aliohjelmaa tai funktiota. Lataus voidaan vielä jaotella ns. karkeajakoiseen lataukseen (Coarse-Grain Binding) ja hienojakoiseen lataukseen (Fine-Grain Binding). Karkeajakoisessa latauksessa valitaan ohjelmamoduuli ja hienojakoisessa latauksessa valitaan ohjelmamoduulissa oleva aliohjelma, funktio, tms.

Kuvassa 1 on pelkistettynä kaaviona esitetty erästä tunnetun tekniikan mukaista latausta. Kutsuva ohjelma A käsittää allohjelmakutsun 1, jossa parametreina annetaan aliohjelman P1, P2 tunniste T1, T2 (tag). Tämä tunniste T1, T2 käsittää tyypillisesti aliohjelman nimen sekä listan

aliohjelmaparametrien tyypeistä. Latauspyyntö välitetään latauskäsittelijälle H, jossa suoritetaan tunnisteen perusteella selvitys, missä ohjelmamoduulissa aliohjelma P1, P2 sijaitsee. Latauskäsittelijä etsii tunnistetietojen perusteella kyseisen ohjelmamoduulin. Tässä etsimisessä voidaan käyttää esimerkiksi hakutaulukkoa (lookup table), jossa on listattuna edullisesti käytettävissä olevat ohjelmamoduulit L1, L2 ja/tai aliohjelman P1, P2 tunnistetiedot T1, T2. Jos latauskäsittelijä löytää sellaisen ohjelmamoduulin L1, L2, jossa kutsuttu aliohjelma P1, P2 sijaitsee, latauskäsittelijä tutkii seuraavaksi sen, vastaavatko aliohjelman tyyppitiedot kutsutun aliohjelman tyyppitietoja. Kuvassa 1 on katkoviivalla 2 merkitty tilannetta, jossa latauskäsittelijä on löytänyt oikean nimen, mutta tyyppitiedot eivät täsmää. Vastaavasti yhtenäisellä nuolella 3 on esitetty tilannetta, jossa sekä nimi- että tyyppitiedot täsmäävät, jolloin latauskäsittelijä siirtää ohjelman toiminnan kyseiseen aliohjelmaan P2 (nuoli 4). Tämä latauskäsittelijä voidaan toteuttaa esimerkiksi käyttöjärjestelmätoimintona tai itse ohjelman yhteydessä.

Edellä kuvattua ohjelmien latausmenetelmää kutsutaan dynaamiseksi lataukseksi erotuksena staattisesta latauksesta, jossa aliohjelmakutsut ja varsinaiset aliohjelmat kytketään jo ohjelman käännösvaiheessa. Staattisella latauksella voidaan jossakin määrin nopeuttaa ohjelman toimintaa, mutta tällöin koko ohjelma tulee ladata tietojenkäsittelylaitteen muistivälineisiin ennen kuin ohjelmaa voidaan käyttää. Myös ohjelman päivitys uudempaan versioon edellyttää koko ohjelman päivittämistä.

Ohjelma ja sen toiminnassa tarvittavat ohjelmamoduulit on yleensä ohjelman asennusvaiheessa tallennettu tietojenkäsittelylaitteen muistivälineisiin, tyypillisesti kiintolevylle. Nykyisin erityisesti verkottumisen lisääntymisen johdosta ei kaikkia ohjelmamoduuleita välttämättä tallenneta samaan tietojenkäsittelylaitteeseen. Esimerkiksi lähiverkossa palvelintietokone sisältää yleensä kaikkien lähiverkossa käytettävissä olevien ohjelmien ohjelmamoduulit, jolloin lähiverkon työasemat lataavat kulloinkin tarvittavat ohjelmamoduulit palvelimen muistivälineistä. Tällaisessakin tilanteessa ohjelmamoduulit on tallennettu tyypillisesti ohjelman asennusvaiheessa palvelimen muistivälineisiin. Internet-tietoverkon suosion lisääntyminen on puolestaan mahdollistanut sen, että

ohjelman ohjelmamoduulit voidaan ladata Internet-verkon välityksellä esimerkiksi ohjelman valmistajan kotisivulta. Tämä lataus voidaan tehdä esim. siinä vaiheessa, kun ohjelma asennetaan työasemalle/palvelimelle, tai siinä vaiheessa, kun ohjelmaa käytetään. Tällöin työasema suorittaa ohjelmamoduulien latauksen Internet-tietoverkon välityksellä esim. ohjelmanvalmistajan kotisivulta. Tällöin työasemassa ei tarvitse olla tallennettuna läheskään kaikkia ohjelmamoduuleita, jolloin muistikapasiteettia esimerkiksi työaseman kiintolevyltä säästyy muuhun käyttöön. Lisäksi ohjelman käytettävissä on aina uusin ohjelmaversio, joten käyttäjän ei välttämättä tarvitse suorittaa ohjelmapäivitystä työasemalla.

Tällaisen dynaamisen ohjelmamoduulin latauksen eräänä epäkohtana on se, että ohjelma voi käyttää sellaistakin ohjelmamoduulia, joka ei ole ohjelman valmistajan tekemä tai kyseisen valmistajan hyväksymän valmistajan toimittama, vaan kyseessä voi olla ns. piraattiversio. Tämän epäkohdan poistamiseksi on kehitetty ns. allekirjoitusmenetelmä, jossa ohjelmamoduuliin lisätään varmenne, jolla ohjelmamoduulin alkuperä pyritään varmistamaan ennen ohjelmamoduulin asennusta. Tämä varmenne on sinänsä tunnetulla salaustekniikalla salattu, jolloin voidaan estää varmenteen väärentäminen. Varmistusta ei kuitenkin tehdä ohjelmaa suoritettaessa, vaan siinä vaiheessa, kun ohjelma asennetaan. Lisäksi varmennuksen suorittaa käyttöjärjestelmä tai muu vastaava työaseman käyttöympäristökomponentti, mutta ohjelma itse ei suorita varmennusta. Tällöin päätöksen ohjelmamoduulin hyväksymisestä tai hylkäämisestä tekee käyttöjärjestelmä. Jos ohjelmamoduuli on työasemassa kerran hyväksytty, voivat kaikki muutkin ohjelmat käyttää kyseistä ohjelmamoduulia. Tämä tilanne voi aiheuttaa ongelmia, koska jonkin muun ohjelman valmistaja ei välttämättä hyväksy tätä ohjelmamoduulia käytettäväksi valmistamansa ohjelman yhteydessä.

Eräänä toisena epäkohtana dynaamisessa latauksessa on se, että eri ohjelmilla voi olla samannimisiä ohjelmamoduuleita ja/tai aliohjelmia. Tunnetun tekniikan mukaisissa työasemissa ei kuitenkaan voi samanaikaisesti olla ladattuna kahta samannimistä ohjelmamoduulia. Tällöin

käyttäjän on huolehdittava ohjelmamoduulin vaihtamisesta siinä tilanteessa, että käyttäjä siirtyy käyttämään toista ohjelmaa, jossa samannimisiä ohjelmamoduuleita käytetään.

Oheisessa kuvassa 2 on esitetty pelkistettynä kaaviona tilannetta, jossa työasemaan on asennettu kaksi ohjelmaa A, B. Ensimmäinen ohjelma A on ensimmäisen valmistajan M1 valmistama ja toimittama. Toinen ohjelma B on toisen valmistajan M2 valmistama ja toimittama. Ensimmäisen ohjelman A tarvitsema ohjelmamoduuli L1 on kolmannen valmistajan M3 valmistama. Ensimmäinen valmistaja M1 on hyväksynyt kolmannen valmistajan M3 toimittamaan kyseisen ohjelmamoduulin L1 käytettäväksi ohjelmassa A. Vastaavasti toinen valmistaja M2 on valtuuttanut neljännen valmistajan M4 toimittamaan toisen ohjelmamoduulin L2 käytettäväksi toisessa ohjelmassa B. Ensimmäinen valmistaja M1 ei kuitenkaan ole valtuuttanut neljättä valmistajaa M4 toimittamaan ohjelmamoduulia L1 omaan ohjelmaansa A. Vastaavasti toisen valmistajan M2 ohjelmassa ei ole valtuutusta kolmannen valmistajan M3 ohjelmamoduulin L1 käyttämiseen, vaikka nämä ohjelmamoduulit L1, L2 sisältäisivätkin olennaisesti saman toiminnallisuuden. Tällöin käyttäjän käynnistäessä ensimmäisen ohjelman A tulee käyttäjän myös ladata ensimmäinen ohjelmamoduuli L1 työasemaan. Lataus suoritetaan tällöin ensimmäisestä ohjelmamoduulista L1, mitä esittää nuoli 5 kuvassa 2. Sen sijaan latausta toisesta ohjelmamoduulista L2 (nuoli 6) ei voi suorittaa. Tilanteessa, jossa käyttäjä siirtyy käyttämään toista ohjelmaa B tulee käyttäjän vaihtaa käytössä oleva ensimmäinen ohjelmamoduuli L1 toiseen ohjelmamoduuliin L2, jotta toinen ohjelma B toimisi halutulla tavalla.

Nyt esillä olevan keksinnön eräänä tarkoituksena on aikaansaada dynaaminen ohjelmamoduulin latausmenetelmä, jolla mahdollistetaan se, että työasemassa voi samanaikaisesti olla käytettävissä useampia samannimisiä ohjelmamoduuleita. Kunkin ohjelman tarvitsema ohjelmamoduuli tunnistetaan ja varmennetaan lisätunnisteen avulla. Nyt esillä olevan keksinnön mukaiselle menetelmälle on tunnusomaista se, mitä on esitetty oheisen patenttivaatimuksen 1 tunnusmerkkiosassa. Nyt esillä olevan keksinnön mukaiselle työasemalle on tunnusomaista se, mitä on esitetty oheisen patenttivaatimuksen 7 tunnusmerkkiosassa.

Keksintö perustuu siihen ajatukseen, että ohjelmamoduuliin lisätään lisätunniste, jonka avulla voidaan haluttu ohjelmamoduuli tunnistaa ja lisäksi varmentaa se, että kyseessä on ohjelman valmistajan tai valtuutetun valmistajan toimittama ohjelmamoduuli.

Nyt esillä olevalla keksinnöllä saavutetaan merkittäviä etuja tunnetun tekniikan mukaisiin ratkaisuihin verrattuna. Keksinnön mukainen menetelmä mahdollistaa kahden tai useamman samannimisen ohjelmamoduulin yhtäaikaisen käytön, jolloin käyttäjän ei tarvitse suorittaa ohjelmamoduulin vaihtotoimenpiteitä. Tämä ohjelmamoduulien yhtäaikainen käyttö on mahdollista toteuttaa jopa siten, että käyttäjän ei edes tarvitse tietää tällaisesta toiminnosta. Lisäksi keksinnöllä saavutetaan se etu, että ohjelmamoduulin toimittaja voidaan varmentaa, jolloin vältytään epäluotettavien ohjelmatoimittajien ohjelmamoduulien käyttämiseltä. Varmennus suoritetaan kutsuvassa ohjelmassa tai sen käynnistämänä, joten varmennus on käyttöympäristöstä riippumaton toimenpide.

Keksintöä selostetaan seuraavassa tarkemmin viitaten samalla oheisiin piirustuksiin, joissa

kuva 1 esittää pelkistettynä kaaviona tunnetun tekniikan mukaista ohjelmamoduulin latausta,

kuva 2 esittää pelkistettynä kaaviona tilannetta tunnetun tekniikan mukaisessa työasemassa kahden ohjelmamoduulin käytön yhteydessä,

kuva 3 esittää pelkistettynä kaaviona keksinnön erään edullisen suoritusmuodon mukaista menetelmää sovellettuna työasemassa,

kuva 4 esittää pelkistetysti lataustoimintoa keksinnön erään edullisen suoritusmuodon mukaisen menetelmän yhteydessä,

kuva 5 esittää periaatekuvaa WAP-mallista, ja

kuva 6 esittää pelkistettynä kaaviona keksinnön erään edullisen suoritusmuodon mukaista päätelaitetta.

WAP (Wireless Application Protocol) on WAP Form -yhteenliittymän määrittämä järjestely Internet-tietoverkon ja kehittyneiden datapalveluiden saannin toteuttamiseksi langattomissa päätelaitteissa. WAP tarjoaa periaatteessa skaalautuvan ja laajennettavissa olevan kokonaisuuden, jonka kerrosrakenteisessa arkkitehtuurissa tietty yhteyskäytäntökerros tarjoaa palveluita seuraavalle kerrokselle. WAP-arkkitehtuuri on hyvin lähellä Internet-tietoverkosta tunnettua WWW-mallia, mutta siihen on tehty langattoman ympäristön vaatimia optimointeja ja muutoksia.

Oheisessa kuvassa 5 on esitetty periaatekuva WAP-mallista, joka mahdollistaa asiakastyöaseman MT ja palvelimen S välisen neuvottelun palvelimella S käytettävän dataobjektin tarjoamiseksi lukijalle ymmärrettävässä muodossa. Asiakastyöasema MT lähettää langattoman tiedonsiirtoverkon NW1 yli yhdyskäytävälle GW koodatun palvelupyynnön, jonka yhdyskäytävä GW dekoodaa ja välittää Internet-tietoverkon NW2 välityksellä palvelimelle S. Palvelin S lähettää pyydetyn sisällön yhdyskäytävälle GW, joka koodaa sisällön ja lähettää sen palvelupyynnön tehneelle asiakkaalle. Vastaanotettu dataobjekti on tulostettavissa käyttäjän tarkasteltavaksi asiakkaan yhteydessä olevan käyttöliittymän välityksellä.

Kuva 6 havainnollistaa pelkistettynä lohkokaaviona asiakastyöasemana käytettävää langatonta päätelaitetta MT. Nyt esitetyssä suoritusmuodossa päätelaitteena on käytetty langatonta viestintä, kuten matkaviestintä, mutta on selvää, että keksintöä voidaan soveltaa myös muuntyyppisten työasemien ja langattomien päätelaitteiden yhteydessä. Langaton viestin voi olla mikä tahansa langaton tietoliikenneväline, kuten esimerkiksi kaksisuuntainen hakulaite, langaton PDA-laite (Personal Digital Assistant), IP-protokollaa käyttävä WLAN-päätelaite (Wireless Local Area Network) tai kannettava tietokone, joka on varustettu laitteistoporttiin lisättävällä antennin käsittävällä matkaviestinverkkokortilla.

Kuvan 6 lohkokaaviossa esitetty langaton viestin sisältää radioteitse tapahtuvaa kommunikointia varten radioyksikön RF, joka käsittää tavanomaisesta matkaviestimestä tunnetun lähetinhaaran TX (käsittäen kanavakoodauksen, lomituksen, salauksen, moduloinnin ja lähetyksen suorittavat toimintolohkot), vastaanotinhaaran RX (käsittäen vastaanoton, demoduloinnin, salauksen purun, lomituksen purun sekä kanavadekoodauksen suorittavat toimintolohkot), radioteitse tapahtuvaa lähetystä varten vastaanoton ja lähetyksen erottavan duplex-suodattimen DF ja antennin ANT. Päätelaitteen toimintaa kokonaisuudessaan ohjaa keskusyksikkö CTRL, joka myös toteuttaa päätelaitteen yhteyskäytännön mukaiset toiminnallisuudet. Matkaviestin käsittää muistin MEM, joka sisältää edullisesti haihtuvaa ja haihtumatonta muistia, ja liitäntäyksikön IO, joka käsittää yhden tai useampia laitteistoportteja sisäisten tai ulkoisten lisälaitteiden liittämiseksi matkaviestimeen. Käyttäjän kanssa suoritettavaa kommunikointia varten työasema käsittää käyttöliittymän, joka sisältää tyypillisesti näppäimistön, näytön, mikrofonin ja kaiuttimen. Tietojenkäsittelyohjelmien yhteydessä liitäntäyksikkö käsittää tiedonsiirtovälineet tietojen siirtämiseksi tietojenkäsittelylaitteen, kuten kannettavan tietokoneen, ja matkaviestimen välillä. Nämä tietojenkäsittelytoiminnot voivat olla toteutettuina myös matkaviestimessä, esimerkiksi kommunikaattorityyppisessä laitteessa, jolloin osa päätelaitteen toiminnoista voi olla yhteisiä sekä matkaviestimelle että tietojenkäsittelytoiminnoille. Yhteys palvelimeen toteutetaan radioyksikön välityksellä. Keskusyksikkö ohjaa matkaviestintoimintojen toteutusta suorittamalla laitteen muistiin ohjelmallisesti tai laitteistorakenteeseen järjestetyt toiminnot, ja edullisesti palvelimelta päätelaitteeseen ladatun ohjelmakoodin toiminnot.

Ohjelmamoduulien sisäinen rakenne voi käytännön sovelluksissa vaihdella. Eräässä edullisessa vaihtoehdossa on aliohjelmien tunnisteille varattu tietty, ennalta määrätty alue. Tällöin sopivimmin näiden tunnisteiden yhteydessä on myös tieto siitä, mistä kohdasta ohjelmamoduulia alkaa aliohjelman ohjelmakoodi. Eräässä toisessa edullisessa vaihtoehdossa kunkin aliohjelman tunniste on aliohjelman ohjelmakoodin yhteydessä. Tällöin tunnisteiden yhteydessä on vielä tieto siitä, missä

kohdassa ohjelmamoduulia sijaitsee seuraavan aliohjelman tunnistetieto. Nyt esillä olevan keksinnön kannalta ei sinänsä ole merkitystä sillä, miten tunnisteet ja aliohjelmat on järjestetty ohjelmamoduuleihin.

Kuvataan seuraavaksi keksinnön erään edullisen suoritusmuodon mukaisen menetelmän toimintaa viitaten samalla kuvaan 3. Käyttäjä käynnistää päätelaitteessa MT moduulirakenteisen ohjelman A pääohjelman. Tämän jälkeen pääohjelmaa suoritetaan, kunnes ohjelman suorituksessa on kutsu 7 aliohjelmaan, joka ei sijaitse samassa ohjelmamoduulissa kuin pääohjelma. Tällöin päätelaitteessa MT siirrytään suorittamaan latauskäsittelijää H, joka voi olla toteutettu esim. suoritettavana olevassa ohjelmassa, tai päätelaitteen MT käyttöjärjestelmätoimintona. Aliohjelmakutsun 7 yhteydessä latauskäsittelijälle H välitetään tieto kutsuttavan aliohjelman nimestä, aliohjelmaparametreista sekä lisätunnisteesta. Latauskäsittelijä käyttää näitä tietoja kutsuttavan aliohjelman P1, P2, P3 selvittämiseksi ohjelmamoduuleihin tallennettujen aliohjelmien tunnisteista T1, T2, T3 edullisesti seuraavasti.

Latauskäsittelijä H suorittaa tunnisteessa T1, T2, T3 olevan ohjelmamoduulin nimen perusteella kyseisen ohjelmamoduulin etsimisen tutkimista varten, kuten on sinänsä tunnettua. Sen jälkeen kun ohjelmamoduuli on löydetty, vertaa latauskäsittelijä ennalta määritetystä paikasta yhden aliohjelman tunnistetietoja aliohjelmakutsussa välitettyihin tietoihin. Tässä vaiheessa tutkitaan edullisesti aliohjelman nimi ja parametrien tyyppitiedot. Mikäli sekä nimi että tyyppitiedot täsmäävät, suoritetaan vielä keksinnön mukaisen lisätunnisteen tutkiminen. Tämän lisätunnisteen perusteella voidaan päätellä, onko tutkittavana oleva ohjelmamoduuli suoritettavana olevan ohjelman valmistajan hyväksymä. Tutkimisessa voidaan käyttää eri menetelmiä, joista erästä kuvataan tarkemmin jäljempänä tässä selityksessä. Näitä lisätunnisteita voi samassa ohjelmamoduulissa olla useampiakin kuin yksi, jolloin useampi valmistaja voi hyväksyä saman ohjelmamoduulin. Kutsuva ohjelma voi välittää myös oman funktionsa, jolla varmistus lisätunnisteen perusteella voidaan suorittaa.

Jos lisätunnisteen tutkimisvaiheessa todetaan, että kyseessä on sellaisen valmistajan toimittama ohjelmamoduuli, jota voidaan käyttää suoritettavana olevan ohjelman yhteydessä, siirrytään suorittamaan tätä aliohjelmaa. Jos sensijaan todetaan, että kyseinen ohjelmamoduuli ei ole sellaisen valmistajan toimittama, joka on suoritettavana olevan ohjelman valmistajan valtuuttama (nuoli 8), siirrytään tutkimaan toista samannimistä ohjelmamoduulia, mikäli sellainen löytyy. Tästä toisesta ohjelmamoduulista tutkitaan, sisältääkö se kutsuttua aliohjelmaa, jossa parametrien tyyppitiedot täsmäävät. Jos tällainen aliohjelma löytyy, tutkitaan vielä ohjelmamoduulin sisältämät lisätunnisteet ohjelmamoduulin valmistajan selvittämiseksi. Jos lisätunniste ei täsmää, kyseessä ei ole valtuutetun valmistajan ohjelmamoduuli (nuoli 9). Jos sen sijaan jokin tutkituista lisätunnisteista osoittaa kyseessä olevan valtuutetun valmistajan toimittama ohjelmamoduuli (nuoli 10), siirrytään suorittamaan tätä aliohjelmaa (nuoli 11), jota kuvassa 3 on merkitty viitteellä P3.

Edellä esitettyjä vaiheita toistetaan edullisesti niin kauan, kunnes haluttu ohjelmamoduuli löytyy, tai kunnes kaikki tietyn nimiset ohjelmamoduulit on käyty läpi. Mainittakoon tässä yhteydessä vielä se, että ohjelmamoduulia voidaan etsiä useista eri paikoista. Etsiminen voidaan aloittaa esimerkiksi päätelaitteen muistista MEM. Tämän jälkeen etsintää voidaan laajentaa lähiverkon NW1 muistivälineisiin, mikäli päätelaite on kytketty tiedonsiirtoyhteyteen johonkin lähiverkkoon. Ohjelmamoduulia voidaan etsiä vielä Internet-tietoverkosta sinänsä tunnetusti. Langattoman päätelaitteen MT yhteydessä voidaan etsimisessä ja latauksessa edullisesti soveltaa mainittua WAP-protokollaa.

Seuraavassa kuvataan tarkemmin eräs edullinen lisätunnisteen toteutusmuoto viitaten samalla kuvan 4 pelkistettyyn kaavioon. Tässä toteutuksessa käytetään digitaalista allekirjoitusta, joka muodostetaan esim. asymmetrisellä salauksella. Asymmetrinen salaus perustuu salainen avain—julkinen avain -avainpariin. Kullakin asymmetristä salausta soveltavalla ohjelmavalmistajalla M1, M2, M3 on yksi tai useampi salainen avain. Näitä on kuvassa 4 merkitty viitteillä SK1, SK2 ja SK3. Kuvassa 4 on selvyyden vuoksi esitetty vain kolme valmistajaa ja jokaiselle valmistajalle yksi avainpari, mutta on selvää, että käytännössä valmistajia

ja avainpareja on huomattavasti enemmän kuin tässä esitetyt. Kuvan 4 esimerkissä on ensimmäinen valmistaja M1 liittänyt ensimmäiseen ohjelmamoduuliin L1 lisätunnisteen, joka käsittää ensimmäisellä salaisella avaimella SK1 muodostetun digitaalisen allekirjoituksen. Tätä lisätunnistetta on esimerkinomaisesti havainnollistettu viitteellä LT1 oheisessa kuvassa 3, mutta on selvää, että käytännön sovelluksissa lisätunnisteen muoto voi vaihdella. Toinen valmistaja M2 on liittänyt toiseen ohjelmamoduuliin L2 lisätunnisteen LT2, joka käsittää toisella salaisella avaimella SK2 muodostetun digitaalisen allekirjoituksen. Myös kolmas valmistaja M3 on vielä liittänyt tähän toiseen ohjelmamoduuliin L2 lisätunnisteen LT3, joka käsittää kolmannella salaisella avaimella SK3 muodostetun digitaalisen allekirjoituksen. Tämä toinen ohjelmamoduuli L2 on siis sekä toisen M2 että kolmannen valmistajan M3 hyväksymä. Kolmas valmistaja M1 on liittänyt myös kolmanteen ohjelmamoduuliin L3 lisätunnisteen LT3, joka käsittää kolmannella salaisella avaimella SK3 muodostetun digitaalisen allekirjoituksen. Tämän kolmannen ohjelmamoduulin L3 tunniste T3 on sama kuin toisen ohjelmamoduulin L2 tunniste T2. Lisätunnisteiden osalta erona on siis se, että kolmannen ohjelmamoduulin L3 lisätunniste ei sisällä toisen valmistajan digitaalista allekirjoitusta.

Päätelaitteen MT käyttäjä on tallentanut päätelaitteeseen tai kuhunkin ohjelmaan, jossa nyt esillä olevaa keksintöä sovelletaan, valmistajan/valmistajien julkisia avaimia PK1, PK2, PK3. Julkisen avaimen jakelu voidaan järjestää esim. siten, että käytetään luotettavaksi arvioitua jakeluorganisaatiota ORG, jolle valmistajat ilmoittavat julkisen avaimensa ja josta käyttäjä voi ne saada. Julkisella avaimella voidaan suorittaa edellä mainittu lisätunnisteen tutkiminen, eli varmistaa se, että digitaalinen allekirjoitus on todella muodostettu julkista avainta vastaavalla salaisella avaimella. Julkiset avaimet tallennetaan siten, että ohjelmaa suoritettaessa ne ovat ohjelman käytettävissä. Kunkin latauksen yhteydessä valitaan julkisista avaimista sellaiset, joita vastaavat valmistajat ovat hyväksyneet ladattavan ohjelmamoduulin käytön. Esimerkiksi kuvan 4 mukaisessa tilanteessa valitaan vain kolmannen valmistajan M3 julkinen avain PK3.

Eräänä vaihtoehtona julkisten avaimien tallennukseen on ns. älykortti (ei esitetty). Älykortti liitetään tiedonsiirtoyhteyteen päätelaitteeseen MT esim. liityntäväylän IF (kuva 5) avulla, jolloin päätelaite MT voi tarvittaessa käydä lukemassa älykorttiin tallennettuja julkisia avaimia.

Digitaalinen allekirjoitus voidaan muodostaa edullisesti sinänsä tunnetulla RSA-algoritmilla, jossa allekirjoitus toteutetaan salausoperaatiolla. Tunnetaan myös muita menetelmiä, kuten DSA (Digital Signature Algorithm), ECDSA (Elliptic Curve Digital Signature Algorithm), joissa digitaalinen allekirjoitus tehdään toisin.

Nyt esillä olevaa keksintöä voidaan soveltaa sellaisissa ohjelmointiympäristölssä, joissa dynaaminen lataus on käytössä. Esimerkkejä tällaisista ympäristöistä mainittakoon tässä yhteydessä CORBA ja Java. Käytännön sovelluksissa keksinnön mukaisen menetelmän yksityiskohdat riippuvat sovellusympäristöstä. CORBA-ohjelmointiympäristössä on käytettävissä bind()-funktio dynaamista latausta varten. Tällöin funktiokutsun yhteydessä voidaan välittää tunniste ja lisätunniste. Bind()-funktion kutsuminen aikaansaa latauskäsittelijän (binding server) suorittamisen, jolloin latauskäsittelijä voi tutkia kutsun yhteydessä välitettyä informaatiota.

Java-ohjelmointiympäristössä käytetään latauksen yhteydessä menetelmien tunnuksia, jotka on muodostettu aliohjelmien nimistä sekä parametrien tyypeistä. Näitä tunnuksia tutkitaan viimeistään siinä vaiheessa, kun lataus on suoritettava. Tällöin tutkimisen suorittaa ns. latauskäsittelijää H vastaavassa Java virtuaalikoneen toiminnallisessa osassa. Tällöin keksinnön mukaista menetelmää voidaan Java-ympäristössä soveltaa esim. siten, että lisätään tunnukseen lisätunniste ja muodostetaan Java virtuaalikoneeseen toiminto, jolla tarkistetaan lisätunnisteen oikeellisuus.

Nyt esillä olevaa keksintöä ei ole rajoitettu ainoastaan edellä esitettyihin suoritusmuotoihin, vaan sitä voidaan muunnella oheisten patenttivaatimusten pulitteissa.

L2

Patenttivaatimukset:

1. Menetelmä ohjelmamoduulin (L1, L2 L3) lataamiseksi päätelaitteessa (MT), jossa suoritetaan yhtä tai useampaa ohjelmaa (A, B), ja jossa menetelmässä aliohjelmia (P1, P2, P3) tallennetaan mainittuihin ohjelmamoduuleihin (L1, L2 L3), ohjelmamoduuleihin (L1, L2 L3) muodostetaan ensimmäisiä tunnistetietoja (T1, T2, T3), jolloin latauksen käynnistämiseksi ohjelmassa suoritetaan kutsu (7) aliohjelmaan (P1, P2, P3), ja kutsuun (7) liitetään ensimmäisiä kutsutietoja (T1, T2, T3) sen ohjelmamoduulin (L1, L2 L3) valitsemiseksi latausta varten, johon kutsuttava aliohjelma (P1, P2, P3) on tallennettu, **tunnettu** siitä, että tunnistetietoihin (T1, T2, T3) liitetään toiset tunnistetiedot (LT1, LT2, LT3), että kutsuun (7) liitetään lisäksi mainittuja toisia kutsutietoja (PKx, PKy, PKz), ja että latauksen yhteydessä verrataan ohjelmamoduuleihin tallennettuja mainittuja ensimmäisiä tunnistetietoja (T1, T2, T3) kutsussa (7) välitettyihin ensimmäisiin kutsutietoihin (T1, T2, T3) ja toisia tunnistetietoja (LT1, LT2, LT3) kutsussa (7) välitettyihin toisiin kutsutietoihin (PKx, PKy, PKz), jolloin ladattavaksi ohjelmamoduuliksi valitaan ohjelmamoduuli, joka vastaa kutsussa välitettyjä ensimmäisiä (T1, T2, T3) ja toisia kutsutietoja (PKx, PKy, PKz).

2. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että ohjelmamoduuleihin muodostettavat toiset tunnistetiedot (LT1, LT2, LT3) sisältävät digitaalisen allekirjoituksen.

3. Patenttivaatimuksen 2 mukainen menetelmä, **tunnettu** siitä, että toisiin kutsutietoihin liitetään julkinen avain (PKx, PKy, PKz), jonka perusteella ohjelmamoduuliin muodostettujen toisten tunnistetietojen digitaalinen allekirjoitus tarkistetaan.

4. Patenttivaatimuksen 1, 2 tai 3 mukainen menetelmä, **tunnettu** siitä, että ohjelmamoduuleihin muodostettavat toiset tunnistetiedot (LT1, LT2, LT3) tallennetaan salatussa muodossa.

5. Patenttivaatimuksen 4 mukainen menetelmä, **tunnettu** siitä, että toisiin kutsutietoihin liitetään julkinen avain (PKx, PKy, PKz), jonka perusteella ohjelmamoduuliin muodostettujen toisten tunnistetietojen (LT1, LT2, LT3) salaus puretaan.

6. Jonkin patenttivaatimuksen 1—5 mukainen menetelmä, jossa ohjelmamoduuleita (L1, L2 L3) tallennetaan Internet-tietoverkkoon tiedonsiirtoyhteydessä olevaan palvelimeen, **tunnettu** siitä, että päätelaitteena (MT) käytetään langatonta päätelaitetta, ja että ohjelmamoduulien (L1, L2 L3) lataus suoritetaan ainakin osittain WAP-protokollan mukaisilla sanomilla.

7. Päätelaite (MT), joka käsittää välineet (H) ohjelmamoduulin (L1, L2 L3) lataamiseksi, joihin ohjelmamoduuleihin (L1, L2 L3) on tallennettu aliohjelmia (P1, P2, P3), ja ensimmäisiä tunnistetietoja (T1, T2, T3), ja joka päätelaite (MT) käsittää lisäksi välineet (CTRL, MEM) ohjelmien (A, B) suorittamiseksi, välineet (CTRL) latauksen käynnistämiseksi suorittamalla ohjelmassa kutsu (7) aliohjelmaan (P1, P2, P3), johon kutsuun (7) on liitetty ensimmäisiä kutsutietoja (T1, T2, T3) sen ohjelmamoduulin (L1, L2 L3) valitsemiseksi latausta varten, johon kutsuttava aliohjelma (P1, P2, P3) on tallennettu, **tunnettu** siitä, että ohjelmamoduuleihin (L1, L2 L3) on tallennettu toisia tunnistetietoja (LT1, LT2, LT3), että päätelaite käsittää lisäksi välineet (CTRL, MEM) toisten kutsutietojen (PKx, PKy, PKz) liittämiseksi kutsuun (7), välineet (H) ohjelmamoduuleihin tallennettujen mainittujen ensimmäisten tunnistetietojen (T1, T2, T3) vertaamiseksi kutsussa (7) välitettyihin ensimmäisiin kutsutietoihin (T1, T2, T3), välineet (H) toisten tunnistetietojen (LT1, LT2, LT3) vertaamiseksi kutsussa (7) välitettyihin toisiin kutsutietoihin (PKx, PKy, PKz), ja välineet (H) ohjelmamoduulin valitsemiseksi ladattavaksi mainittujen vertailujen perusteella.

8. Patenttivaatimuksen 7 mukainen päätelaite (MT), **tunnettu** siitä, että ohjelmamoduuleihin muodostetut toiset tunnistetiedot (LT1, LT2, LT3) sisältävät digitaalisen allekirjoituksen.

9. Patenttivaatimuksen 8 mukainen päätelaite (MT), **tunnettu** siitä, että toisiin kutsutietoihin on liitetty julkinen avain (PKx, PKy, PKz), jonka perusteella ohjelmamoduuliin muodostettujen toisten tunnistetietojen digitaalinen allekirjoitus on järjestetty tarkistettavaksi.

10. Patenttivaatimuksen 7, 8 tai 9 mukainen päätelaite (MT), joka käsittää välineet (RF, DF, ANT) Internet-tietoverkkoon (NW2) tiedonsiirtoyhteydessä olevaan palvelimeen (S) tallennettujen ohjelmamoduuleiden (L1, L2 L3) lataamiseksi päätelaitteeseen (MT), **tunnettu** siitä, että päätelaite (MT) on langaton päätelaite, ja että se käsittää välineet (CTRL) ohjelmamoduulien (L1, L2 L3) latauksen suorittamiseksi ainakin osittain WAP-protokollan mukaisilla sanomilla.

L3

(57) Tiivistelmä:

Keksinnön kohteena on menetelmä ohjelmamoduulin (L1, L2 L3) lataamiseksi päätelaitteessa (MT), jossa suoritetaan yhtä tai useampaa ohjelmaa (A, B), jossa menetelmässä aliohjelmia (P1, P2, P3) tallennetaan mainittuihin ohjelmamoduuleihin (L1, L2 L3). Ohjelmamoduuleihin (L1, L2 L3) muodostetaan ensimmäisiä tunnistetietoja (T1, T2, T3), jolloin latauksen käynnistämiseksi ohjelmassa suoritetaan kutsu (7) aliohjelmaan (P1, P2, P3). Kutsuun (7) liitetään ensimmäisiä kutsutietoja (T1, T2, T3) sen ohjelmamoduulin (L1, L2 L3) valitsemiseksi latausta varten, johon kutsuttava aliohjelma (P1, P2, P3) on tallennettu. Tunnistetietoihin liitetään toiset tunnistetiedot (LT1, LT2, LT3). Kutsuun (7) liitetään lisäksi mainittuja toisia kutsutietoja (PKx, PKy, PKz). Latauksen yhteydessä verrataan ohjelmamoduuleihin tallennettuja mainittuja ensimmäisiä tunnistetietoja (T1, T2, T3) kutsussa (7) välitettyihin ensimmäisiin kutsutietoihin (T1, T2, T3) ja toisia tunnistetietoja (LT1, LT2, LT3) kutsussa (7) välitettyihin toisiin kutsutietoihin (PKx, PKy, PKz), jolloin ladattavaksi ohjelmamoduuliksi valitaan ohjelmamoduuli, joka vastaa kutsussa välitettyjä ensimmäisiä (T1, T2, T3) ja toisia kutsutietoja (PKx, PKy, PKz). Keksinnön kohteena on myös päätelaite (MT).

Fig. 4

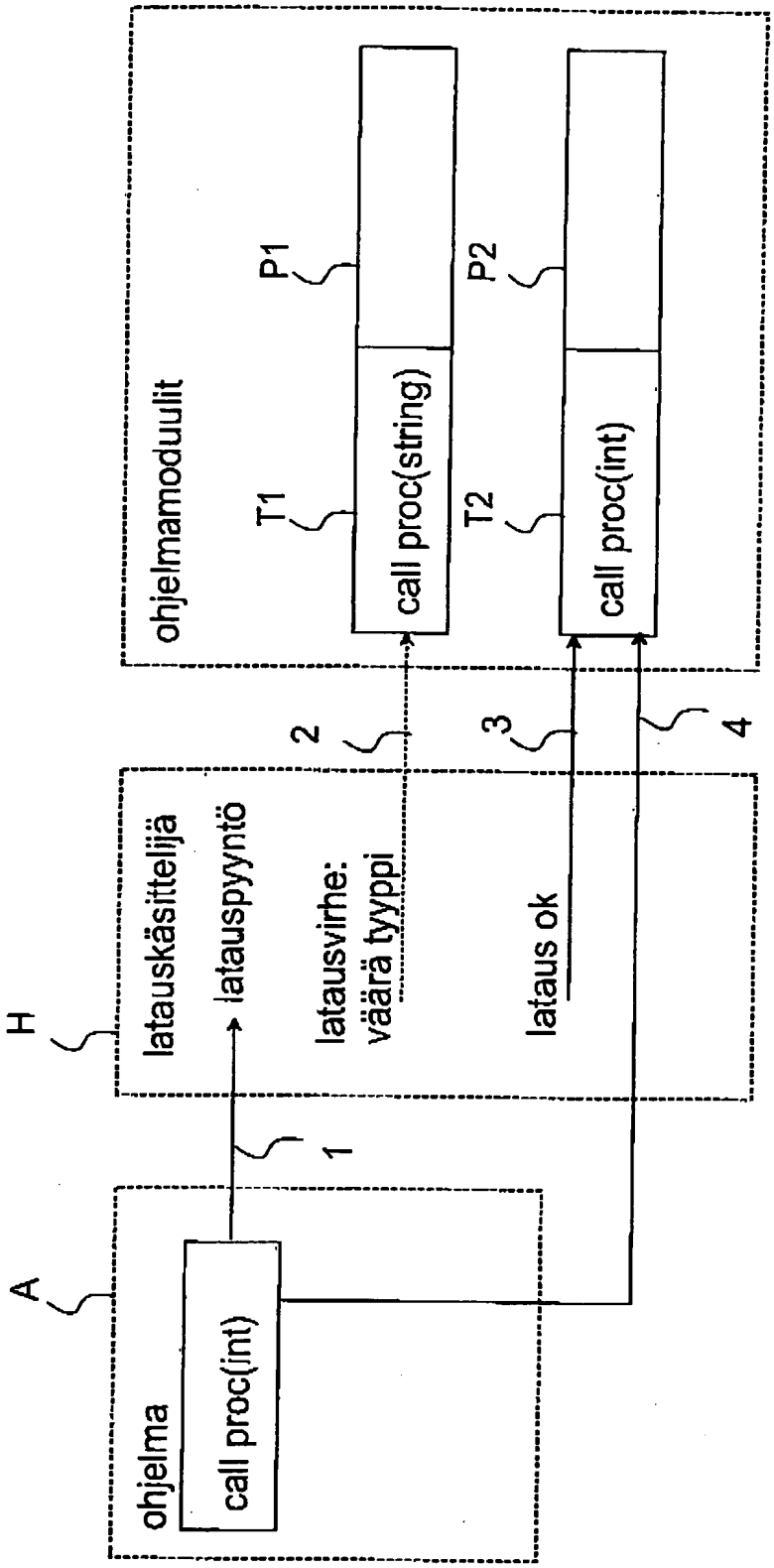
ohjelmamoduulit
P1
P2
T1
T2
call proc(string)
call proc(int)
2
3
4
H
latauskäsittelijä
latauspyyntö
latausvirhe:
väärä tyyppi
lataus ok
1
A
ohjelma
call proc(int)

Fig. 1

2

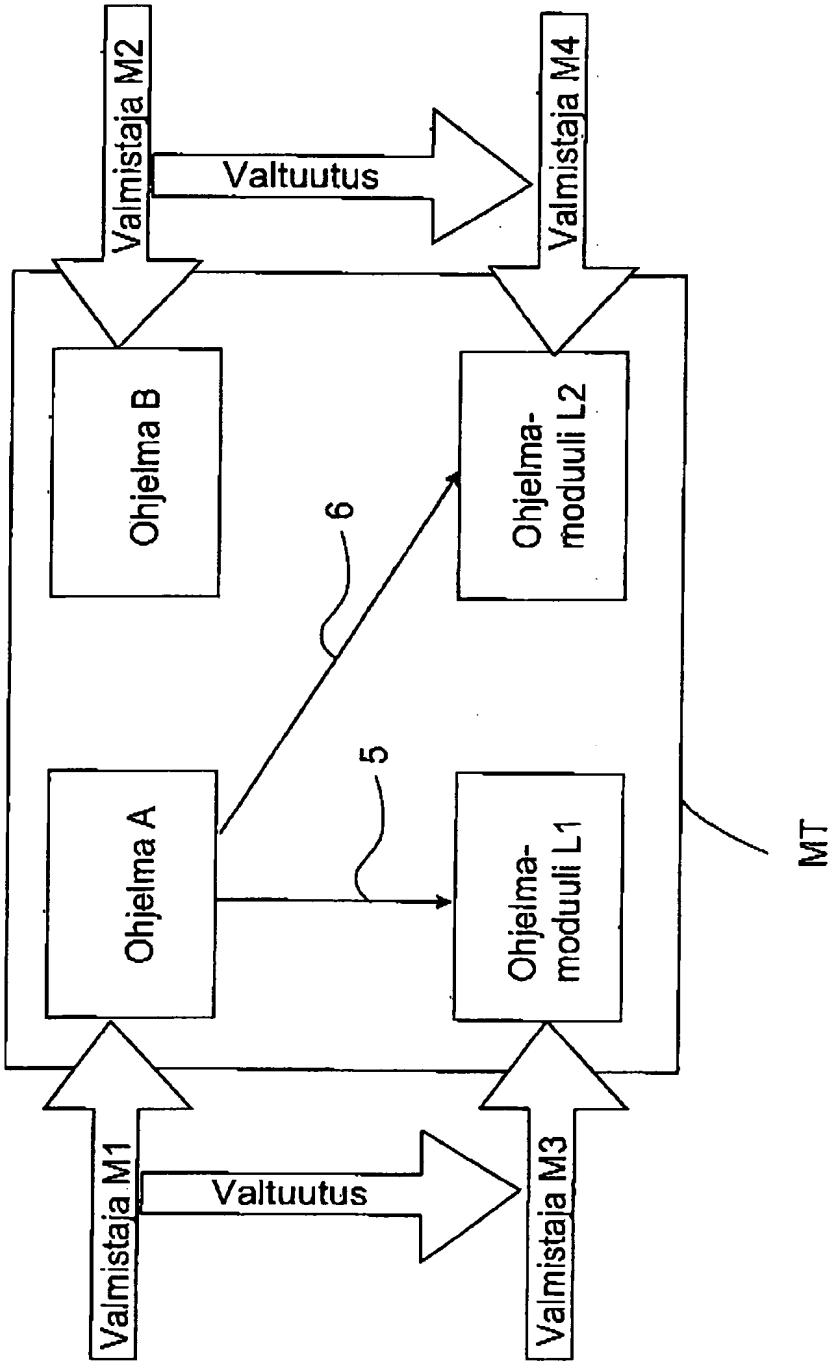
Valmistaja M2
Valtuutus
Valmistaja M4
Ohjelma B
Ohjelma-moduuli L2
6
5
Ohjelma A
Ohjelma-moduuli L1
MT
Valmistaja M1
Valtuutus
Valmistaja M3

Fig. 2

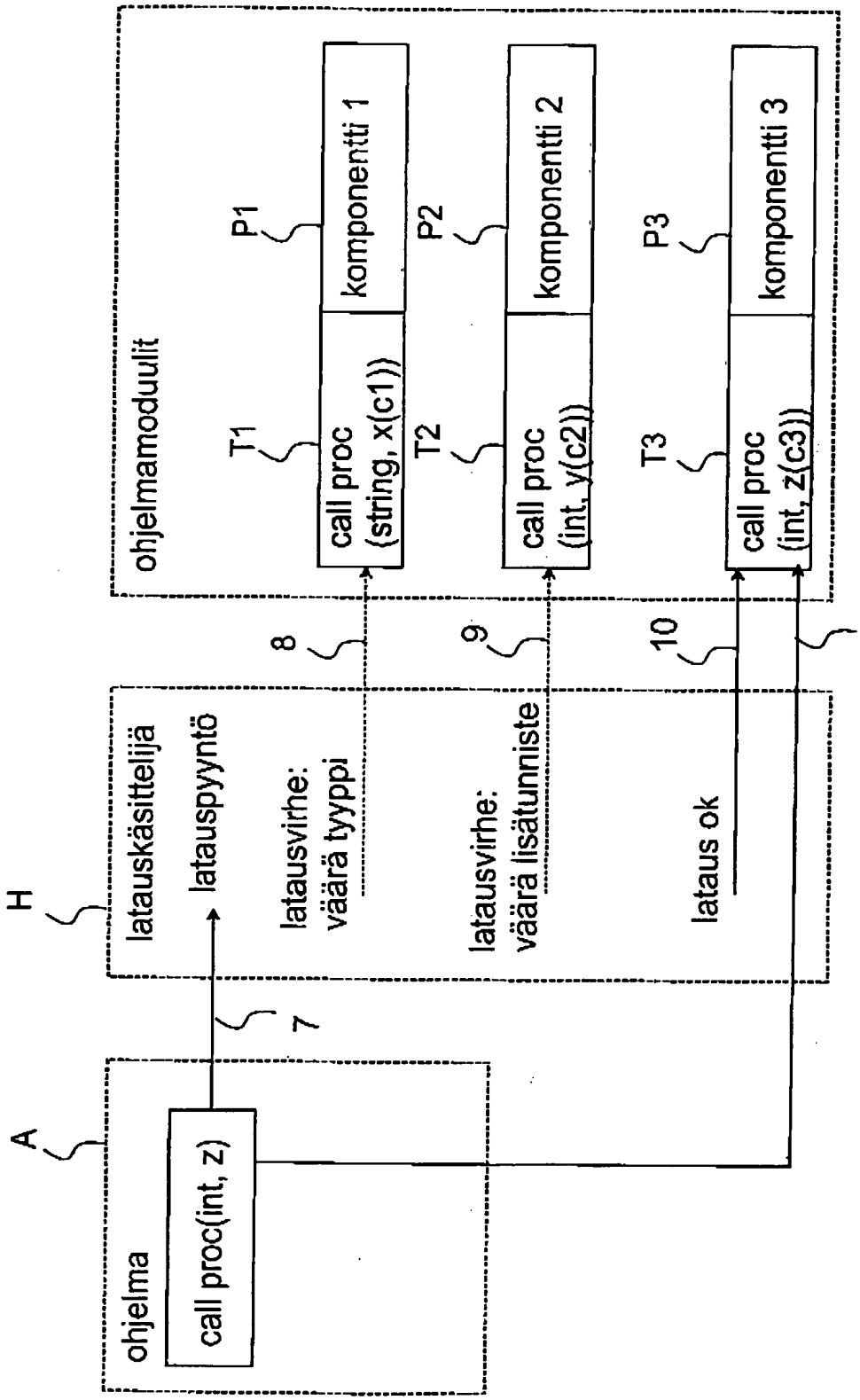
ohjelma
call proc(int, z)
A
7
H
latauskäsittelijä
latauspyyntö
latausvirhe:
väärä tyyppi
latausvirhe:
väärä lisätunniste
lataus ok
8
9
10
11
ohjelmamodulit
T1
call proc
(string, x(c1))
P1
komponentti 1
T2
call proc
(int, y(c2))
P2
komponentti 2
T3
call proc
(int, z(c3))
P3
komponentti 3

Fig. 3

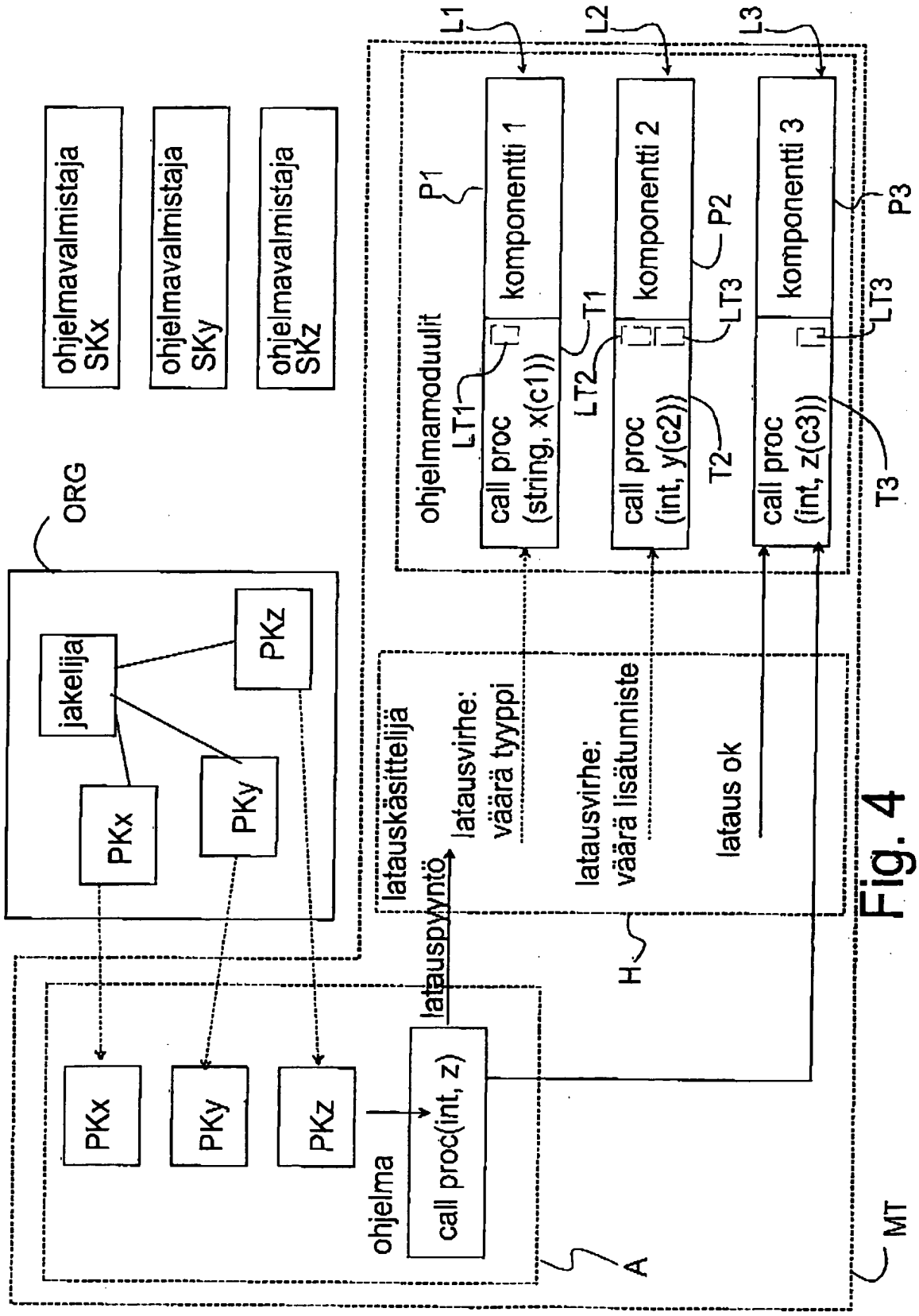
ohjelmavalmistaja SKx
ohjelmavalmistaja SKy
ohjelmavalmistaja SKz
ORG
jakelija
PKx
PKy
PKz
ohjelmamoduulit
P1
P2
P3
komponentti 1
komponentti 2
komponentti 3
L1
L2
L3
LT1
LT2
LT3
T1
T2
T3
call proc (string, x(c1))
call proc (int, y(c2))
call proc (int, z(c3))
latauskäsittelijä
latausvirhe: väärä tyyppi
latausvirhe; väärä lisätunniste
lataus ok
latauspyyntö
H
MT
A
ohjelma
call proc(int, z)
PKx
PKy
PKz

Fig. 4

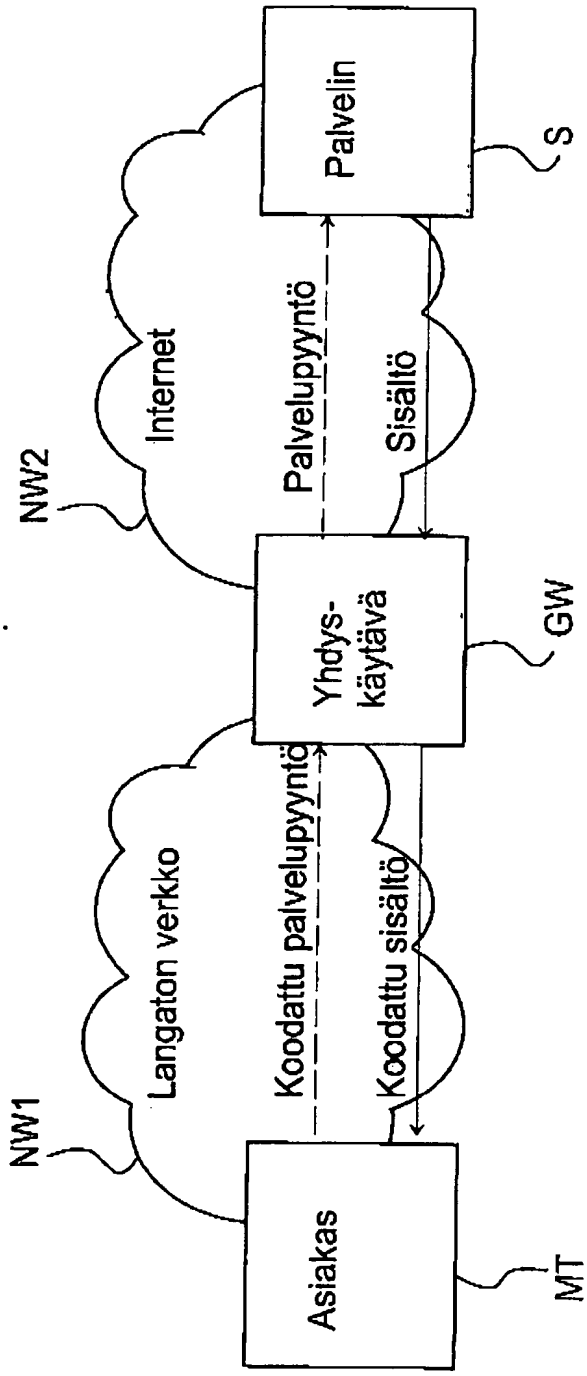
Palvelin
S
Internet
Palvelupyyntö
Sisältö
NW2
Yhdys-
käytävä
GW
Langaton verkko
Koodattu palvelupyyntö
Koodattu sisältö
NW1
Asiakas
MT

Fig. 5

6

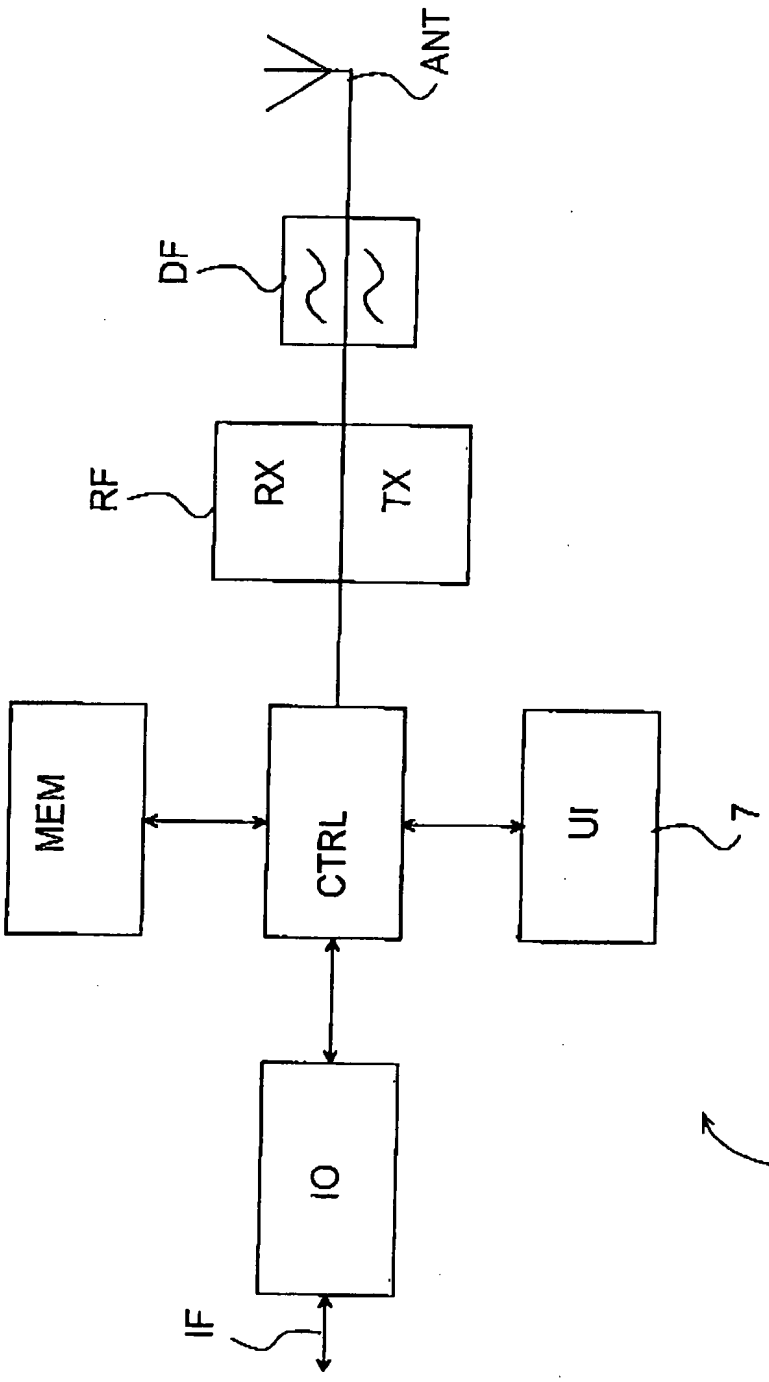

Fig. 6

CERTIFICATE

I, Tuulikki Tulivirta, hereby certify that, to the best of my knowledge and belief, the following is a true translation, for which I accept responsibility, of a certified copy of Finnish Patent Application 19992786 filed on 27 January 1999.

Tampere, 15 December 2000

Tuulikki Tulivirta

Tuulikki Tulivirta
Certified Translator (Act 1148/88)

Tampereen Patenttitoimisto Oy
Hermiankatu 6
FIN-33720 TAMPERE
Finland

Method for binding a program module

The present invention relates to a method for binding a program module as set forth in the preamble of the appended claim 1. The invention also relates to a terminal as set forth in the preamble of the appended claim 7.

At present, programs are primarily implemented as having a module structure, wherein one program consists of several program modules, modules, *etc.* It is thus possible to keep the size of a single program module within limits, and during the use, only such program modules are bound which are used at a time. Such an arrangement facilitates *e.g.* the maintenance of the programs, reduces the memory capacity required in the data processor using the program, as well as makes it possible for several different programs to use the same program module. To run such a program consisting of modules, a so-called main program is started. When the operation of the main program proceeds, for example on the basis of instructions entered by users, the program module needed at the time is bound from the main program. Below in this specification, the general name program module will be used to refer to such a program component, in connection with which binding can be applied. Such program modules include *e.g.* program libraries.

Binding refers in this specification to a process related to running of programs, in which a program component, such as a main program or a subroutine, calls another program component, such as a subroutine or a function. Binding can be further divided into so-called coarse-grain binding and fine-grain binding. In coarse-grain binding, a program module is selected, and in fine-grain binding, a subroutine, function or the like is selected from the program module.

Figure 1 shows, in a reduced chart, a way of binding according to prior art. A calling program A comprises a subroutine call 1, in which a tag T1, T2 for the subroutine P1, P2 is given as the parameters. This tag T1, T2 typically comprises the name of the subroutine and a list of the types of the subroutine parameters. The binding request is transferred to a binding server H, where the tag is used to find out in which program module the subroutine P1, P2 is located. In this search, it is

possible to use *e.g.* a lookup table which preferably comprises a list of the available program modules L1, L2 and/or the tags T1, T2 of the subroutine P1, P2. If the binding server finds such a program module L1, L2, in which the called subroutine P1, P2 is located, the binding server H will next examine, if the type data of the subroutine correspond to the type data of the called subroutine. In Fig. 1, a broken line 2 indicates a situation, in which the binding server H has found a correct name but the type data do not match. In a corresponding manner, a solid arrow 3 is used to indicate a situation, in which both the name and type data match, wherein the binding server H hands the operation of the program over to said subroutine P2 (arrow 4). This binding server can be implemented *e.g.* as an operating system function or in connection with the program itself.

The above-described method for binding programs is called dynamic binding, as distinguished from static binding, where subroutine calls and the actual subroutines are linked already at the stage of compiling the program. Static binding can accelerate the operation of the program to some extent, but in this case the whole program must be loaded into the memory means of the data processor before the program can be used. Moreover, the updating of the program to a newer version requires that the whole program is updated.

The program and the program modules required in its operation are usually stored at the stage of installing the program on the memory means of the data processor, typically on a fixed disk. At present, particularly due to an increase in the use of networks, all program modules are not necessarily stored in the same data processor. For example in a local area network, a server computer normally comprises the program modules of all the programs available in the local area network, wherein work stations of the local area network load the program modules needed at the time from the memory means of the server. Also in such a situation, the program modules are typically stored at the stage of installing the program in the memory means of the server. On the other hand, the increased popularity of the Internet data network has made it possible to download the program modules of the program by means of the Internet network *e.g.* from the home page of the program manufacturer. This downloading can be performed *e.g.* at the

stage when the program is installed in a work station or a server, or at the stage when the program is used. Thus, the work station performs the downloading of the program modules via the Internet data network *e.g.* from the home page of the program manufacturer. Thus, not nearly all the program modules need to be stored in the work station, wherein storage capacity *e.g.* on the fixed disk of a work station is left over for other use. Furthermore, the newest program version is always available for the program, wherein the user does not necessarily need to perform updating of the program at the work station.

One disadvantage in binding of such a dynamic program module is that the program can use even such a program module which is not made by the program manufacturer or is not supplied by a manufacturer approved by the manufacturer in question, but it can be a so-called pirate version. To eliminate this drawback, a so-called signature method has been developed, in which the program module is supplemented with a confirmation which is used as an attempt to verify the origin of the program module before installing program module. This confirmation is encrypted with an encryption method known *per se*, to prevent falsification of the confirmation. However, the confirmation is not made upon running the program but at the stage of installing the program. Furthermore, the confirmation is made by the operating system or another corresponding component in the use environment of the work station, but not by the program itself. Thus, the decision on accepting or rejecting the program module is made by the operating system. If the program module has been once accepted at the work station, also all the other programs can use the program module in question. This situation may cause problems, because the manufacturer of another program may not necessarily accept this program module to be used in connection with a program made by itself.

Another drawback in dynamic binding is that different programs may have program modules and/or subroutines having the same name. In work stations of prior art, however, two program modules with the same name cannot be loaded simultaneously. Thus, the user must take care of changing the program module in a situation when the user switches over to use another program using program modules with the same name.

The appended Fig. 2 shows, in a reduced chart, a situation in which two programs A, B are installed in a work station. The first program A is manufactured and supplied by a first manufacturer M1. The second program B is manufactured and supplied by a second manufacturer M2. A program module L1 necessary for the first program A is manufactured by a third manufacturer M3. The first manufacturer M1 has authorized the third manufacturer M3 to supply said program module L1 to be used in the program A. In a corresponding manner, the second manufacturer M2 has authorized a fourth manufacturer M4 to supply a second program module L2 to be used in the second program B. However, the first manufacturer M1 has not authorized the fourth manufacturer M4 to supply the program module L1 in its own program A. In a corresponding manner, the program by the second manufacturer M2 has no authorization to use the program module L1 of the third manufacturer M3, even if these program modules L1, L2 contained substantially the same functionality. Thus, when the user starts the first program A, the user must also load the first program module L1 in the work station. The binding is thus performed from the first program module L1, which is indicated by an arrow 5 in Fig. 2. Nevertheless, binding from the second program module L2 (arrow 6) cannot be performed. In a situation in which the user switches over to use the second program B, the user must also change the first program module L1 in use to the second program module L2 so that the second program B would run in the desired manner.

It is an aim of the present invention to provide a dynamic method for binding a program module, to make it possible to have several program modules with the same name available for use simultaneously in a work station. The program module required by each program is identified and verified by means of an auxiliary tag. The method according to the present invention is characterized in what will be presented in the characterizing part of the appended claim 1. The work station according to the present invention is characterized in what will be presented in the characterizing part of the appended claim 7. The invention is based on the idea that the program module is supplemented with an auxiliary tag, which can be used to identify a desired program module and also to

verify that it is a program module supplied by the program manufacturer or by an authorized manufacturer.

Significant advantages are achieved by the present invention when compared with solutions of prior art. The method according to the invention makes it possible to use two or more program modules with the same name simultaneously, wherein the user does not need to take any measures to change the program module. This simultaneous use of program modules can even be implemented in such a way that the user does not even need to know about such a function. Furthermore, the invention has the advantage that the supplier of the program module can be confirmed, whereby the use of program modules by unreliable program suppliers can be avoided. The confirmation is executed in the calling program or is started up by the same, wherein the confirmation is an operation independent of the use environment.

In the following, the invention will be described in more detail with reference to the appended drawings, in which

Fig. 1 shows, in a reduced chart, binding of a program module according to prior art,

Fig. 2 shows, in a reduced chart, the situation in a work station of prior art in connection with the user of two program modules,

Fig. 3 shows, in a reduced chart, the method according to a preferred embodiment of the invention applied in a work station,

Fig. 4 shows, in a reduced manner, the binding function in connection with the method according to a preferred embodiment of the invention,

Fig. 5 shows a WAP model in a skeleton diagram, and

Fig. 6 shows, in a reduced chart, a terminal according to a preferred embodiment of the invention.

WAP (Wireless Application Protocol) is an arrangement determined by the WAP Forum for implementing access to the Internet data network and sophisticated data services in mobile terminals. WAP offers an entity which is, in principle, scalable and expandable, and in whose layer-structured architecture a certain protocol layer offers services for the next layer. WAP architecture is very close to the WWW model known from the Internet data network, but it contains some optimizations and changes required by the wireless environment.

The appended Fig. 5 shows a skeleton diagram of a WAP model which makes communication possible between a client work station MT and a server S, to offer the reader a data object used at the server S in an intelligible format. The client work station MT transmits an encoded service request over a wireless communication network NW1 to a gateway GW, which the gateway GW decodes and transmits via the Internet data network NW2 to the server S. The server S transmits the requested content to the gateway GW, which codes the content and transmits it to the client that made the service request. The received data object can be printed out to be reviewed by the client by means of the user interface in connection with the client.

Figure 6 illustrates, in a reduced block chart, a mobile terminal MT used as a client work station. In the present embodiment, the terminal used is a wireless communication device, such as a mobile station, but it is obvious that the invention can also be applied in connection with other types of work stations and mobile terminals. The mobile station can be any wireless communication means, such as *e.g.* a duplex paging device, a wireless PDA device (Personal Digital Assistant), a WLAN (Wireless Local Area Network) terminal using the IP protocol, or a portable computer equipped with a mobile communication network card to be inserted in the hardware port and comprising an antenna.

The mobile station shown in the block chart of Fig. 6 comprises, for communication via the radio channel, a radio part RF which normally comprises, in a way known from a conventional mobile station, a transmitter branch TX (comprising the functional blocks which perform channel coding, interlacing, encryption, modulation, and transmission),

a receiver branch RX (comprising the functional blocks which perform reception, demodulation, decryption, de-interlacing, and channel decoding), a duplex filter DF for separating reception and transmission, and an antenna ANT for transmission on the radio channel. The operation of the terminal as a whole is controlled by a central unit CRTL which also implements the functionalities complying with the protocol of the terminal. The mobile station comprises a memory MEM, which contains preferably a volatile and a non-volatile memory, and an interface unit IO comprising one or several hardware ports for connecting internal or external auxiliary device sto the mobile station. For communication with the user, the work station comprises a user interface which typically comprises a keypad, a display, a microphone, and a speaker. In connection with data processing programs, the interface unit comprises communication means for transmitting data between the data processor, such as a portable computer, and the mobile station. These data processing functions can also be implemented in the mobile station, for example in a communicator-type device, wherein some of the functions of the terminal can be common to both the mobile station and the data processing functions. The connection with the server is implemented via a radio unit. The central unit controls the implementation of the mobile station functions by performing the functions arranged as software in the memory of the device or in the hardware structure, and preferably the functions of the program code loaded from the server to the terminal.

The internal structure of the program modules can vary in practical applications. In an advantageous alternative, a certain predetermined area is allocated for the tags of the subroutines. Thus, preferably in connection with these tags, there is also information about the location in the program module where the program code of the subroutine starts. In another advantageous alternative, the tag of each subroutine is in connection with the program code of the subroutine. Thus, in connection with the tags, there is also information about the location of the tag of the next subroutine in the program module. In view of the present invention, it is not significant as such, how the tags and subroutines are arranged in the program modules.

We shall next describe the operation of an advantageous embodiment of the invention with reference to Fig. 3. The user starts the main program of a program A with a module structure in a terminal MT. After this, the main program is run, until there is a call 7 in the program running to a subroutine which is not located in the same program module as the main program. Thus, the terminal MT moves on to execute a binding server H which can be implemented *e.g.* in the program being run, or as an operating system function in the terminal MT. In connection with the subroutine call 7, information is transmitted to the binding server H about the name, subroutine parameters and auxiliary code of the subroutine to be called. The binding server uses this information to determine the subroutine P1, P2, P3 to be called from the tags T1, T2, T3 of the subroutines stored in the program modules, preferably in the following way.

On the basis of the name of the program module in the tag T1, T2, T3, the binding server H searches for said program module for examination, which is known *per se*. After the program module has been found, the binding server compares the tag of one subroutine from a predetermined location with the data transmitted in the subroutine call. At this stage, the name of the subroutine and the type data of the parameters are preferably examined. If both the name and the type data match, the auxiliary tag according to the invention is still examined. On the basis of this auxiliary tag, it is possible to determine whether the program module under examination has been approved by the manufacturer of the program being run. In this examination, it is possible to use various methods, one of which will be described in more detail below in this description. The same program module can also contain more than one of these auxiliary tags, wherein several manufacturers can approve the same program module. The calling program can also transmit its own function, whereby the confirmation on the basis of the auxiliary tag can be performed.

If it is found at the stage of examining the auxiliary tag that the program module in question is supplied by such a manufacturer that it can be used in connection with the program to be run, the next step is to run this subroutine. Nevertheless, if it is found that the program module in question is not supplied by a manufacturer authorized by the manufac-

turer of the program to be run (arrow 8), the next step is to examine another program module with the same name, if there is one. This second program module is examined to find out if it contains a called subroutine in which the parameter type data match. If such a subroutine is found, the auxiliary tags contained by the program module are also examined to find out the manufacturer of the program module. If the auxiliary tag does not match, the program module in question is not one by an authorized manufacturer (arrow 9). However, if one of the examined auxiliary tags indicates that the program module in question is one supplied by an authorized manufacturer (arrow 10), the next step is to run this subroutine (arrow 11), which is indicated with the reference P3 in Fig. 3.

The above-presented steps are preferably iterated as long as the desired program module is found, or until all the program modules with a certain name have been examined. In this context, it should also be mentioned that the program module can be searched for in several different locations. For example, the search can be started in the terminal memory MEM. After this, the search can be expanded to the memory means of a local area network NW1, if the terminal is coupled to communicate with a local area network. Furthermore, the program module can be searched for in the Internet network in a way known *per se*. In connection with the wireless terminal MT, said WAP protocol can be preferably applied in the search and the binding.

In the following, an advantageous embodiment of an auxiliary tag will be described in more detail with reference to the skeleton diagram of Fig. 4. In this implementation, a digital signature is used, which is formed *e.g.* by asymmetric encryption. Asymmetric encryption is based on the key pair of a secret key and a public key. Each program manufacturer M1, M2, M3 applying asymmetric encryption has one or more secret keys. These are indicated with the references SK1, SK2 and SK3 in Fig. 4. For clarity, Fig. 4 only shows three manufacturers and one key pair for each manufacturer, but it is obvious that in practice, there are considerably more manufacturers and key pairs than those presented here. In the example of Fig. 4, the first manufacturer M1 has supplemented the first program module L1 with an auxiliary tag which comprises a digital signature formed by a first secret key SK1. As an

example, this auxiliary tag is illustrated with the reference LT1 in the appended Fig. 3, but it is obvious that the format of the auxiliary tag can vary in practical applications. The second manufacturer M2 has supplemented a second program module L2 with an auxiliary tag LT2 which comprises a digital signature formed by a second secret key SK2. Furthermore, a third manufacturer M3 has supplemented this second program module L2 with an auxiliary code LT3 which comprises a digital signature formed by a third secret key SK3. Thus, this second program module L2 is approved by both the second M2 and the third manufacturer M3. The third manufacturer M3 has supplemented also a third program module L3 with the auxiliary tag LT3 which comprises a digital signature formed by the third secret key SK3. The tag T3 of this third program module L3 is the same as the tag T2 of the second program module L2. Consequently, for the auxiliary tags, the difference lies in that the auxiliary tag of the third program module L3 does not contain the digital signature of the second manufacturer.

In the terminal or in each program, in which the present invention is applied, the user of the terminal MT has stored public keys PK1, PK2, PK3 of the manufacturer/manufacturers. The distribution of the public key can be arranged *e.g.* by using a distribution organization ORG which is evaluated as reliable and to which the manufacturers give their public keys and from which the user can obtain them. The public key can be used to perform the above-mentioned examination of the auxiliary tag, *i.e.* to secure that the digital signature is really formed by the secret key corresponding to the public key. The public keys are stored in such a way that they are available for use of the program when the program is being run. In connection with each binding, such keys are selected from the public keys that correspond to the manufacturers which have authorized the use of the program module to be bound. For example, in the situation of Fig. 4, only the public key PK3 of the third manufacturer M3 is selected.

One alternative for storing public keys is a so-called intelligent card (not shown). The intelligent card is connected to communicate with the terminal MT *e.g.* via an interface bus IF (Fig. 6), wherein the terminal MT can read public keys stored on the intelligent card, when necessary.

The digital signature can be preferably formed by an RSA algorithm, known *per se*, in which the signature is implemented by an encryption operation. There are also other known methods, such as the DSA (Digital Signature Algorithm), ECDSA (Elliptic Curve Digital Signature Algorithm), in which the digital signature is performed in another way.

The present invention can be applied in such programming environments which use dynamic binding. Examples of such environments which should be mentioned in this context are CORBA and Java. In practical applications, the details of the method according to the invention depend on the application environment. In the CORBA programming environment, a bind() function is available for dynamic binding. Thus, in connection with the function call it is possible to transmit a tag and an auxiliary tag. Calling the bind() function activates the binding server, wherein the binding server can examine the information transmitted in connection with the call.

In the Java programming environment, in connection with binding, method tags are used which are formed of subroutine titles and parameter types. These tags are examined at the latest at the stage when binding must be performed. Thus, the examination is performed by an operating part of the Java virtual machine corresponding to the binding server H. The method of the invention can thus be applied in the Java environment *e.g.* by supplementing the tag with an auxiliary tag and forming a function in the Java virtual machine to verify the auxiliary tag.

The present invention is not limited solely to the embodiments presented above, but it can be modified within the scope of the appended claims.

Claims:

1. A method for binding a program module (L1, L2, L3) in a terminal (MT), in which one or several programs (A, B) are running, and in which method subroutines (P1, P2, P3) are stored in said program modules (L1, L2, L3), the program modules (L1, L2, L3) are provided with first tags (T1, T2, T3), wherein to start binding, the program makes a call (7) to a subroutine (P1, P2, P3), and the call (7) is supplemented with the first tags (T1, T2, T3) to select the program module (L1, L2, L3) for binding, in which the called subroutine (P1, P2, P3) is stored, **characterized** in that the tags (T1, T2, T3) are supplemented with second tags (LT1, LT2, LT3), that the call (7) is also supplemented with said second call data (PKx, PKy, PKz), and that in connection with the binding, said first tags (T1, T2, T3) stored in the program modules are compared with the first tags (T1, T2, T3) transmitted in the call (7), and the second tags (LT1, LT2, LT3) are compared with the second call data (PKx, PKy, PKz) transmitted in the call (7), wherein the program module to be bound is selected to be the program module which matches with the first tags (T1, T2, T3) and the second call data (PKx, PKy, PKz) transmitted in the call.

2. The method according to claim 1, **characterized** in that the second tags (LT1, LT2, LT3) to be formed in the program modules contain a digital signature.

3. The method according to claim 2, **characterized** in that the second call data are supplemented with a public key (PKx, PKy, PKz), on the basis of which the digital signature of the second call data formed in the program module is verified.

4. The method according to claim 1, 2 or 3, **characterized** in that the second tags (LT1, LT2, LT3) to be formed in the program modules are stored in an encrypted form.

5. The method according to claim 4, **charact riz d** in that the second call data are supplemented with a public key (PKx, PKy, PKz), on the basis of which the second tags (LT1, LT2, LT3) formed in the program modules are decrypted.

6. The method according to any of the claims 1 to 5, in which program modules (L1, L2, L3) are stored in a server communicating with the Internet network, **characterized** in that the terminal (MT) used is a mobile terminal, and that the binding of the program modules (L1, L2, L3) is performed at least partly by messages complying with the WAP protocol.

7. A terminal (MT) comprising means (H) for binding a program module (L1, L2, L3), which program modules (L1, L2, L3) contain stored subroutines (P1, P2, P3) and first tags (T1, T2, T3), and which terminal (MT) also comprises means (CTRL, MEM) for running programs (A, B), means (CTRL) for starting binding by performing in the program a call (7) to a subroutine (P1, P2, P3), the call (7) being supplemented with first call data (T1, T2, T3) to select that program module (L1, L2, L3) for binding in which the called subroutine (P1, P2, P3) is stored, **characterized** in that the program modules (L1, L2, L3) contain stored second tags (LT1, LT2, LT3); that the terminal also comprises means (CTRL, MEM) for adding second call data (PKx, PKy, PKz) to the call (7), means (H) for comparing said first tags (T1, T2, T3) stored in the program modules with the first call data (T1, T2, T3) transmitted in the call (7), means (H) for comparing the second tags (LT1, LT2, LT3) with the second call data (PKx, PKy, PKz) transmitted in the call (7), and means (H) for selecting a program module to be bound on the basis of said comparison.

8. The terminal (MT) according to claim 7, **characterized** in that the second tags (LT1, LT2, LT3) formed in the program modules contain a digital signature.

9. The terminal (MT) according to claim 8, **characterized** in that the second call data are supplemented with a public key (PKx, PKy, PKz), on the basis of which the digital signature of the second tags formed in the program module are arranged to be verified.

10. The terminal (MT) according to claim 7, 8 or 9, comprising means (RF, DF, ANT) for binding program modules (L1, L2, L3) stored in a server (S) communicating with the Internet network (NW2),

charact rized in that the terminal (MT) is a mobile terminal, and that it comprises means (CTRL) for performing binding of the program modules (L1, L2, L3) at least partly by messages complying with the WAP protocol.

Abstract

The invention relates to a method for loading a program module (L1, L2, L3) in a terminal, in which one or several programs (A, B) are running, and in which method subroutines (P1, P2, P3) are stored in said program modules (L1, L2, L3). The program modules (L1, L2, L3) are provided with first tags (T1, T2, T3), wherein to start binding, the program makes a call (7) to a subroutine (P1, P2, P3). The call (7) is supplemented with the first tags (T1, T2, T3) to select the program module (L1, L2, L3) for binding, in which the called subroutine (P1, P2, P3) is stored. The tags (T1, T2, T3) are supplemented with second tags (LT1, LT2, LT3). The call (7) is also supplemented with said second call data (PKx, PKy, PKz). In connection with the binding, said first tags (T1, T2, T3) stored in the program modules are compared with the first tags (T1, T2, T3) transmitted in the call (7), and the second tags (LT1, LT2, LT3) are compared with the second call data (PKx, PKy, PKz) transmitted in the call (7), wherein the program module to be bound is selected to be the program module which matches with the first tags (T1, T2, T3) and the second call data (PKx, PKy, PKz) transmitted in the call. The invention also relates to a terminal (MT).

Fig. 4

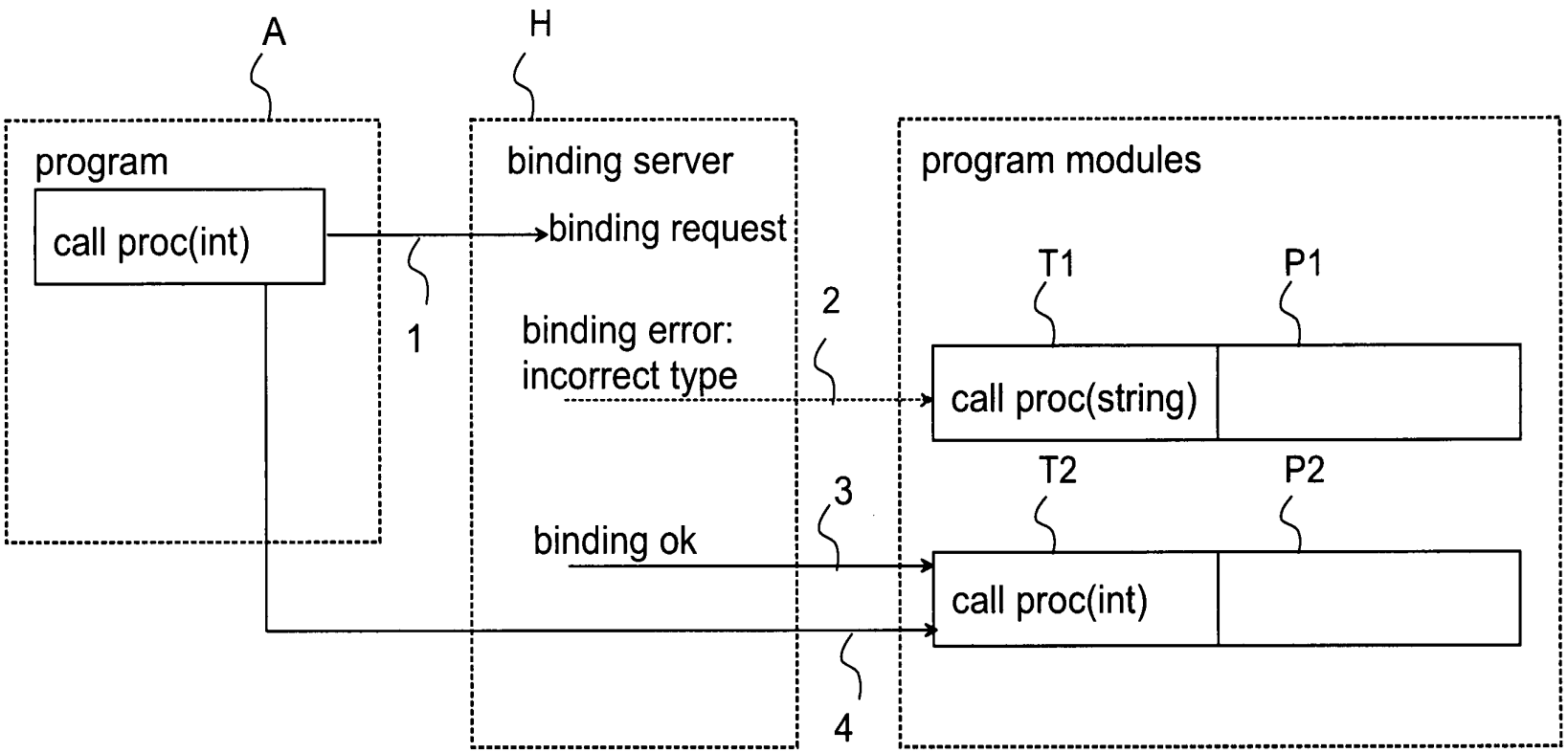
A
H
program
call proc(int)
binding server
binding request
1
binding error:
incorrect type
2
program modules
T1
P1
call proc(string)
T2
P2
3
binding ok
call proc(int)
4

Fig. 1

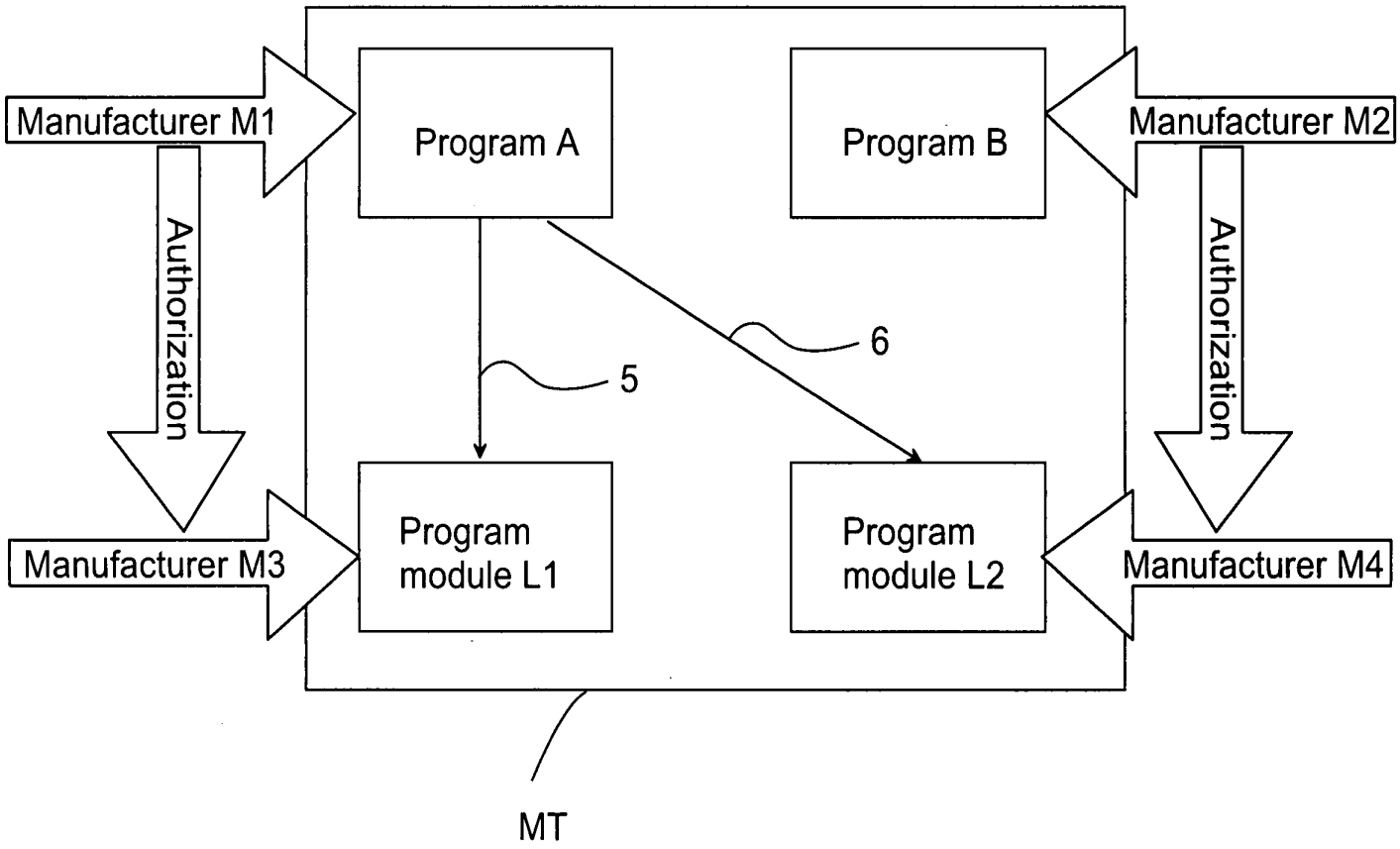
Manufacturer M1
Program A
Program B
Manufacturer M2
Authorization
Authorization
6
5
Manufacturer M3
Program module L1
Program module L2
Manufacturer M4
MT

Fig. 2

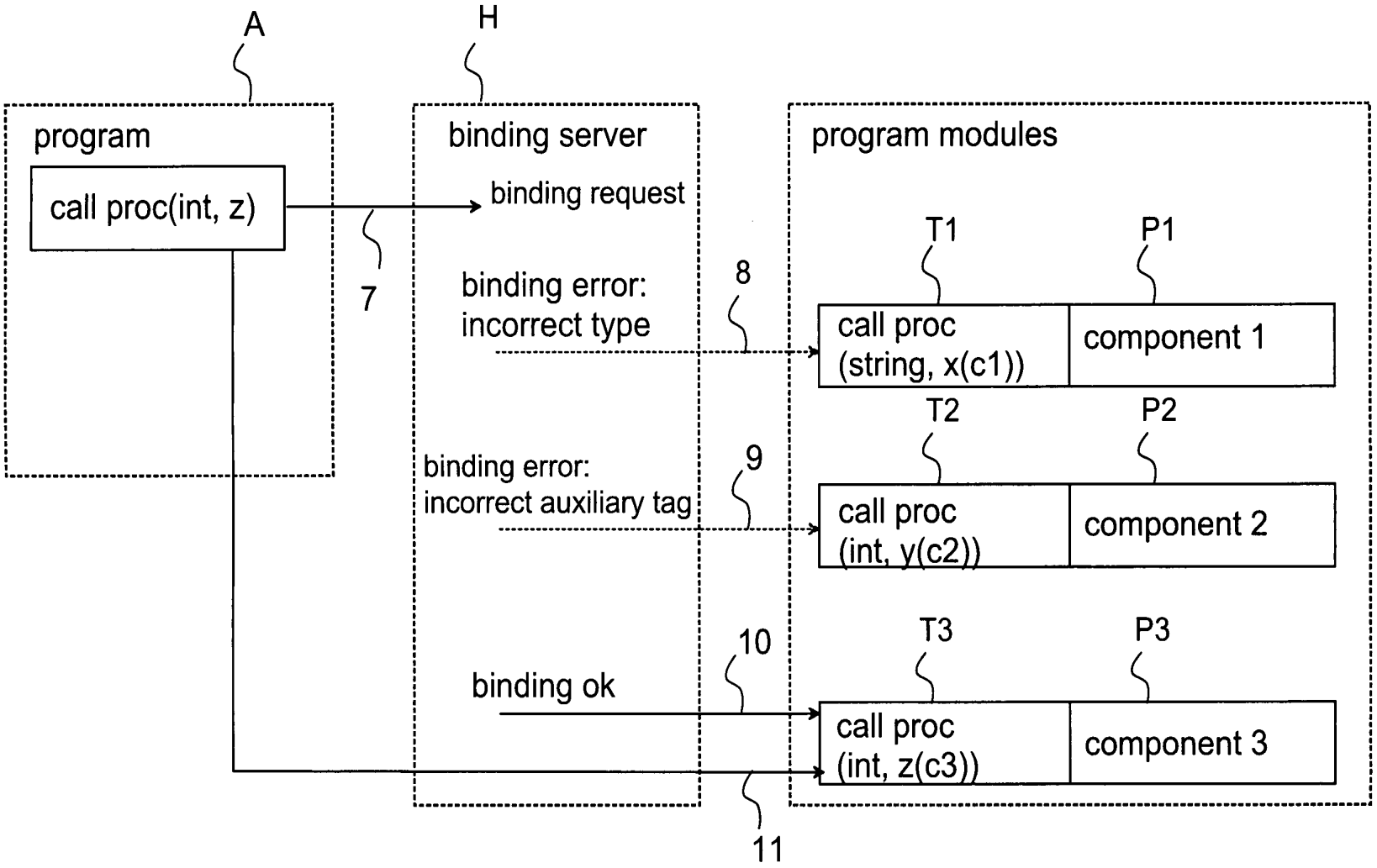
A
H
program
call proc(int, z)
binding server
binding request
7
binding error:
incorrect type
8
binding error:
incorrect auxiliary tag
9
binding ok
10
11
program modules
T1
P1
call proc
(string, x(c1))
component 1
T2
P2
call proc
(int, y(c2))
component 2
T3
P3
call proc
(int, z(c3))
component 3

Fig. 3

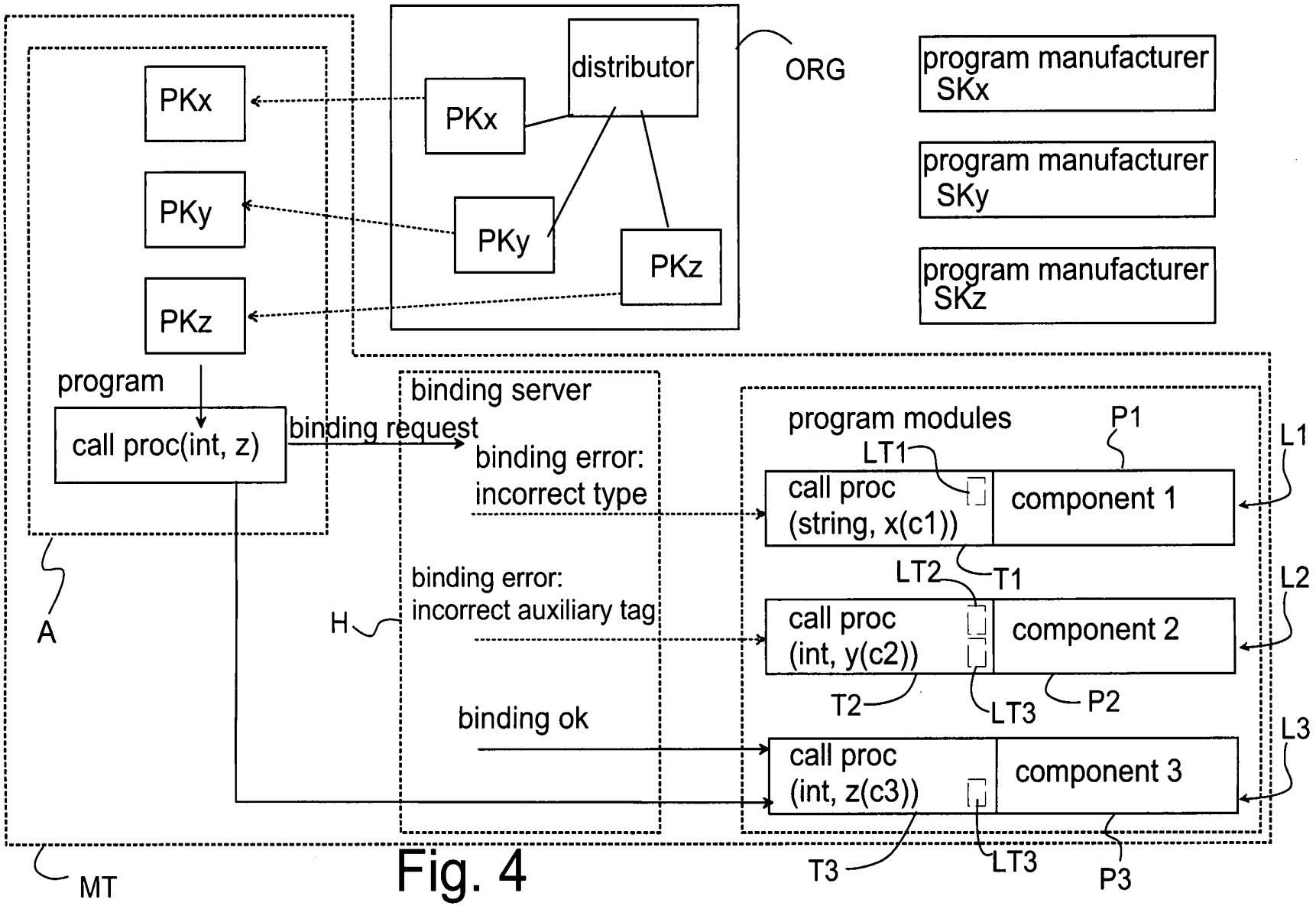
PKx
PKy
PKz
distributor
PKx
PKy
PKz
ORG
program manufacturer SKx
program manufacturer SKy
program manufacturer SKz
program
call proc(int, z)
binding request
binding server
binding error: incorrect type
binding error: incorrect auxiliary tag
binding ok
program modules
P1
LT1
call proc (string, x(c1))
component 1
L1
LT2
T1
call proc (int, y(c2))
component 2
L2
T2
LT3
P2
call proc (int, z(c3))
component 3
L3
T3
LT3
P3
A
H
MT

Fig. 4

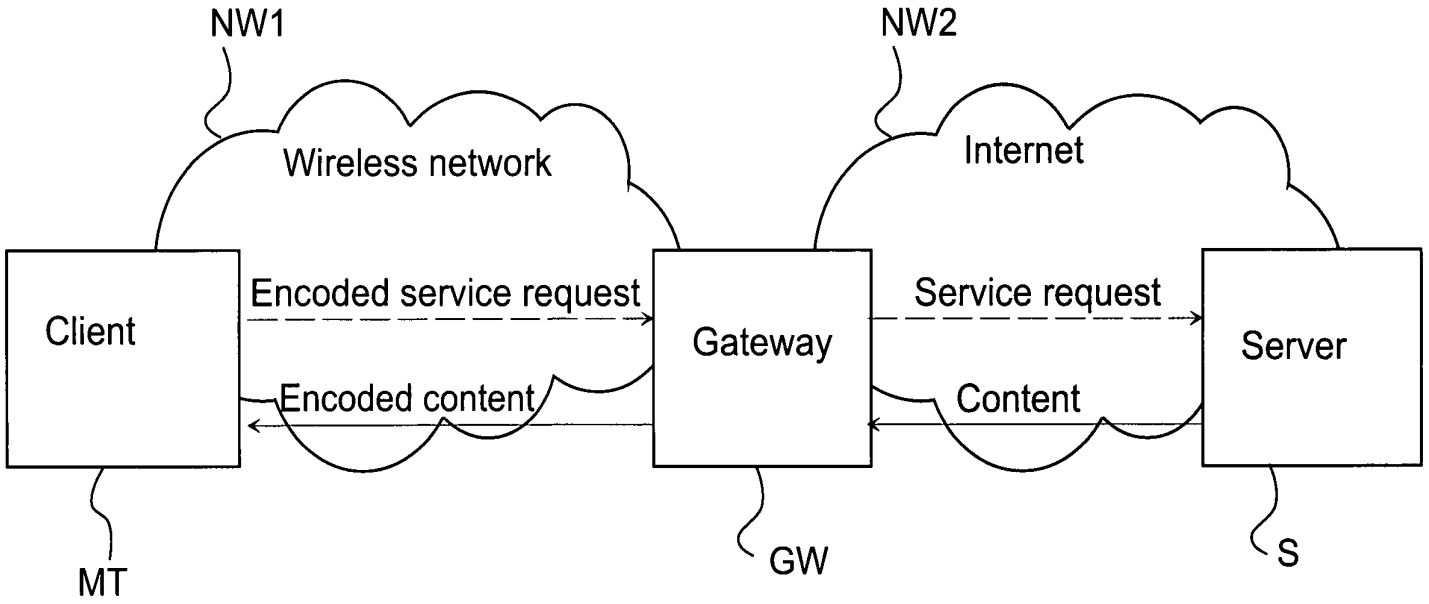
NW1
NW2
Wireless network
Internet
Client
Encoded service request
Gateway
Service request
Server
Encoded content
Content
MT
GW
S

Fig. 5

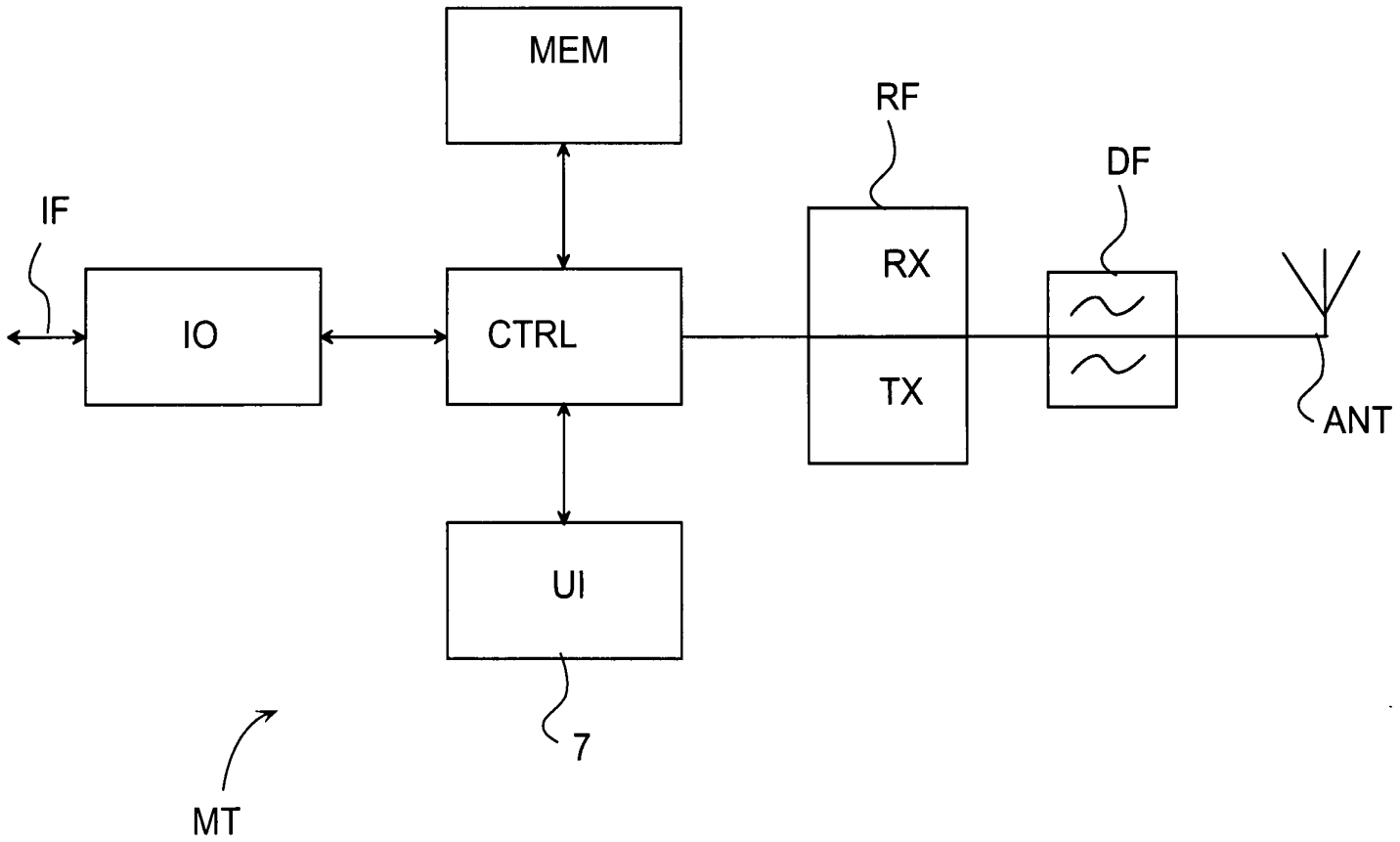

Fig. 6